В.И. Каширин

Программа РОС: расчет и оценка качества оптических систем

Взгляд изнутри

Екатеринбург, 2019

СОДЕРЖАНИЕ

Предисловие

Какую задачу призван решить расчёт оптических систем?

Исходя из заданных критериев, предъявленных заказчиком к качеству изображения, требуется определить состав и значение внутренних параметров оптической системы.

Расчёту оптических систем (далее – ОС) только на русском языке посвящено бессчётное количество учебников, монографий и статей (см., например, список Литературы в Приложении 1). Кстати, любители модерна возмущаются, что в литературных ссылках приводятся работы, датированные «прошлым веком». Необходимо иметь в виду, что именно в пионерских, а также в ранних работах даётся наиболее обстоятельное описание существа рассматриваемых вопросов. Цитирующие или ссылающиеся на эти работы авторы, в целях экономии печатного места, а порой по лености, походя дают куцее, невнятное разъяснение, отчего суть вопросов исчезает, а понимание не образуется.

Каким образом прежде решалась задача расчёта ОС?

Все имеющиеся руководства по расчёту ОС сосредоточены на аналитическом, умозрительном рассмотрении предмета. Это объясняется «духом» той эпохи, когда учёный мир опирался в основном на «мощь» своего интеллекта и когда численные методы исследований ещё только-только входили в практику.

В XIX веке Зейделем был предложен и разработан, а его последователями доведён до совершенства алгебраический метод расчёта ОС, основанный на аналитической теории аберраций 3го порядка. Задача получения заданного качества изображения в этой теории заменена задачей удовлетворения оптической системой искусственным математическим критериям – коэффициентам аберраций 3-го порядка, которым соответствуют идеализированные ошибки ОС: сферическая аберрация, кома, астигматизм, кривизна поля, дисторсия и две хроматические аберрации – хроматизм положения и хроматизм увеличения. Наивысшую завершённость алгебраический метод получил в работе В.Н. Чуриловского [5].

Многие расчётные программы и сегодня следуют так называемому принципу «разделения переменных», согласно которому на первой стадии расчёта параметры ОС определяются и оптимизируются в области Зейделя, и только на последней стадии расчёта система «улучшается тригонометрической подгонкой».

Однако такой подход неизбежно приводит к необходимости работы с системой в диалоговом режиме, с обязательным присутствием разработчика, к привязанности его к разрабатываемой системе. Кроме того, будучи встроенным в общий программный код, блок оптимизации ОС по коэффициентам Зейделя, будучи плодом раздвоения личности, только тормозит движение процесса к оптимуму, поскольку постоянно сбивает оптимизатор на ложный путь, уводя его от точного решения в сторону приближенного.

Электронно-вычислительные машины (ЭВМ), в середине XX века взявшие на себя всю рутинную работу по высокоточным вычислениям, в принципе изменили всю методологию разработки ОС. К тому времени алгоритмы оптимизации сложных систем и процессов достигли высокого уровня совершенства. Поэтому сегодня использование теории Зейделя при расчёте ОС воспринимается как атавизм: быстродействие современных ЭВМ позволяет при тестировании ОС производить прямую оценку её качества по изображению, как искусственных, программно синтезированных объектов, так и по снимкам реальных объектов. Это даёт возможность разработчику, однажды полностью поставив машине задачу, буквально из набора плоскопараллельных пластин получить в итоге все парамеры объектива с ожидаемыми характеристиками.

В имеющейся литературе изложение практических приложений, начиная с прогона лучей через ОС и кончая теорией Зейделя, даётся в незавершённом виде, когда остаточные аберрации высокого порядка приводятся, как досадный диссонанс между стройной теорией Зейделя и результатами геометрического прогона лучей. Поэтому ни в одном руководстве по расчё-

ту ОС нет ни строчки программного кода, предоставляющего исследователю возможность дальнейшего развития и адаптации под свои нужды и предпочтения расчётной программы. Как в своё время умножение столбиком двух чисел считалось творческим процессом, так и программные коды в те годы (да и теперь ещё) относят к так называемому «ноу-хау».

Цель настоящего пособия состоит в том, чтобы на примере анализа исходного кода существующей расчётной программы POC дать начинающему расчётчику внятные разъяснения, почему именно так, а не иначе, в ней ведётся исследование ОС. Это предоставит возможность исследователю ОС произвести самостоятельную адаптацию и развитие программы под свои нужды. Для привития практических навыков по алгоритмизации и программированию решения оптических задач к настоящему пособию прилагается одноимённый Лабораторный практикум.

Таких программ, как POC, в мире разработан уже не один десяток, начиная со знаменитого Земакса и кончая мало кому известным OSODом, написанным на архаичном Фортране. Расчётчик оптики должен уметь не только профессионально пользоваться этими программами, но и модернизировать их, изменяя их код и включая в свою рабочую версию расчётной программы полезные элементы из стороннего софта.

При изложении материала основное внимание уделено моментам, менее всего освещённым в литературе, а потому наиболее трудным для понимания и реализации в коде. Предполагается, что с остальным программным кодом читатель окажется способным разобраться самостоятельно.

ЗАМЕЧАНИЕ.
В Приложениях ссылки на интернет-источники даны по состоянию на 18 июля 2019 года.

Введение

Наверное, не имеет смысла делать широкое и углублённое введение в основы расчёта ОС, поэтому дадим общую канву, так сказать, взгляд на эту область с высоты птичьего полёта.

ОС является сложным объектом проектирования, тысячами нитей связанным с физической оптикой, технологией, математикой, эргономикой и пр., пр. В таких случаях используются приёмы математического моделирования объектов и процессов. Во время своей работы расчетные программы производят численные эксперименты над математической моделью. В основе каждого такого эксперимента лежит тест ОС посредством прогона пучков лучей через неё.

Учёт явления оптической дифракции максимально приближает исследование качеств оптики к физической реальности и представляет собой самостоятельную проблему. Волновая ошибка, составляющая зрачковую функцию системы, естественным образом напрямую связана с дифракцией и сопряжёнными с нею функциями качества: функцией рассеяния точки (ФРТ) и частотно-конрастной фарактеристикой (ЧКХ). Именно зрачковая функция содержит в себе всю информацию об ОС и её потенциальных возможностях по обеспечению качества изображения.

Будучи плавно изменяемой, зрачковая функция ОС может быть посчитана с достаточной степенью точности для большинства ОС, что обеспечивает надёжное основание для объективной оценки качества системы. При дальнейших препарированиях зрачковой функции указанное информационное содержание сохраняется благодаря адекватности процедуры преобразования Фурье. Быстрое преобразование Фурье (БПФ) – это не абстрактный аппарат по учёту статистики в понятиях синус-косинус, а уникальное средство тончайшего математического анализа системы без явного взятия производных по зрачковой функции.

Столь же таинственной представляется начинающему расчётчику и технология расшифровки интерферограмм с целью оценки уровня волновых аберраций в исследуемой ОС.

Особую роль и значение приобретают полиномы Цернике при расчёте ОС и, особенно, при регрессионном анализе интерферограмм, позволяющем реконструировать зрачковую функцию исследуемой ОС, т.е. восстановить фазу и размах волновой ошибки по форме полос на интерферограмме. Следует подчеркнуть, что полиномы Цернике, являясь совершенно незаменимым средством при регрессионном восстановлении зрачковой функции, при расчёте ОС выступают в качестве навязанного протеза: быстродействия современных ЭВМ вполне достаточно для того, чтобы в реальном времени производить тестирование реальной ОС, а не заменять её харектеристики искусственным полиномным суррогатом.

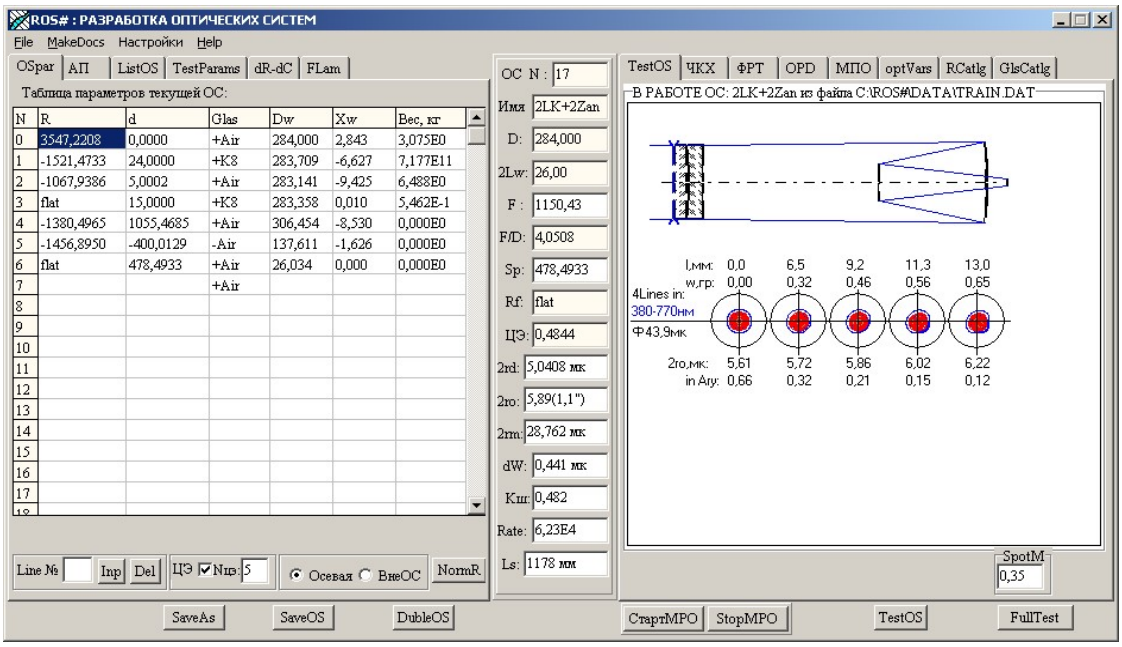
I. ПРОГРАММА РОС

Программа для исследования ОС создавалась не одно десятилетие.

Начало программе положено в 1977м, когда потребовалось определять продольные аберрации параболоида Ф180/5 при его изготовлении и контроле методом Фуко из центра кривизны.

Из-за отсутствия вычислительной техники первые ОС приходилось исследовать на клавишных калькуляторах, раскладывая в ряды синусы и косинусы. Затем, в 1980м, появился программируемый микрокалькулятор МК3-34, а в 1980х уже использовались «Искра», ДВК, РК-84. В 1990х появился «Спектрум» Клайва Синклера, и, наконец, в 1991м возникла IBM-386.

Программирование велось в машинных кодах, на Ассемблере, затем – на Бейсике, Паскале и на Си. Были перепробованы различные среды программирования: Борланд Паскаль (и Си), Визуальный Бейсик (и Си), а также Дельфи. Ближе всех по духу и объективной целесообразности пришлась среда Дельфи.



Главное окно программы РОС.

В 2004м программа РОС была оформлена на Паскале как самостоятельная программа для полноценной работы под ДОС, а в 2010м она была переведена под Windows XP-SP3 с помощью среды программирования Delphi-7-Lite.

В 2013 РОС подверглась фундаментальной переработке и по функциональным возможностям была доведена до уровня, необходимого для профессиональной работы: к программе были подключены отечественные и зарубежные каталоги стекол; показатели преломлений стекол стали браться не из таблиц, а рассчитываться по 13 аппроксимирующим формулам; была установлена связь с программой ZeMax; осознание роли зрачковой функции позволило ввести в программу тесты по сложным объектам – нескольким звездам, мИрам и снимкам реальных объектов. Последнее потребовало введения в программу алгоритмов обработки цифровых снимков и новых критериев оценки качества фотографических ОС (астрографов).

Проблема повышения разрешающей способности у полиапертурного телескопа (ПАТ) потребовала развития программы для работы с внеосевыми ОС.

Заманчивой оказалась идея создания безобективного телескопа, регистрирующего с помощью матриц ПЗС непосредственно волновой фронт света, излучаемого объектом. Был введён блок чтения фотоснимков, полученных с помощью «Светоскопа» (Lite-Scop).

1. СОСТАВ ПРОЕКТА РОС И ПЕРЕЧЕНЬ МОДУЛЕЙ

Проект состоит из нескольких файлов разного функционального назначения.

Головным файлом является ROS.DPR – программа, организующая, запускающая и координирующая работу всего проекта.

Вот содержание этого файла:

```
program ROS;

uses
  Forms,
  Windows,
  Dialogs,
  main in 'main.pas' {frmMain},
  ROSsets in 'ROSsets.pas' {SetsForm},
  ShowLOS in 'ShowLOS.pas' {frmLOSshow},
  viewAPVP in 'viewAPVP.pas' {frmViewAPVP},
  ShowHelp in 'ShowHelp.pas' {frmHelp},
  About in 'About.pas' {aboutForm},
  FullTestOS in 'FULLTestOS.PAS' {frmFullTestOS},
  AllObjTest in 'AllObjTest.pas' {frmAllObjTest},
  Base in 'Base.pas',
  ClcOS in 'ClcOS.pas',
  CtgRR in 'CtgRR.pas',
  CtgGls in 'CtgGls.pas',
  SvLd in 'SvLd.PAS',
  OptUn in 'OptUn.PAS',
  WrkOS in 'WrkOS.PAS',
  GrROS in 'GrROS.pas',
  ChrOD in 'ChrOD.pas',
  Testing in 'Testing.pas',
  Utils in 'UTILS.PAS',
  PFwrk in 'PFwrk.pas',
  Face in 'Face.pas',
  FitUn in 'FitUn.pas',
  TstImg in 'TstImg.pas',
  WrkZmx in 'WrkZmx.pas';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfrmMain, frmMain);
  Application.Run;
end.
```

Модули Forms, Windows и Dialogs являются инструментальными средствами среды программирования Дельфи и обеспечивают работоспособность программы РОС в операционной среде Windows: создают окна, обеспечивают им связь с операционной системой и аппаратурой компьютера.

Следующие 8 модулей, также разработанные при непосредственном содействии программной среды Дельфи, формируют рабочие окна программы РОС (в скобках указаны названия «форм», присвоенные разработчиком):

'main.pas' {frmMain} – основное окно программы РОС,
'ROSsets.pas' {SetsForm} – окно с основными уставками программы,
'ShowLOS.pas' {frmLOSshow} – окно просмотра списка ОС,
'viewAPVP.pas' {frmViewAPVP} – окно для просмотра свойств асферических поверхностей,
'ShowHelp.pas' {frmHelp} – окно со справочной информацией о программе РОС,
'About.pas' {aboutForm} – завстака (рекламное окно «О программе»),
'FULLTestOS.PAS' {frmFullTestOS} – окно «Полный тест ОС»,
'AllObjTest.pas' {frmAllObjTest} – окно дифракционного тестирования ОС по всем искусственным объектам.

Далее перечислены все рабочие модули программы, написанные её разработчиком на языке Паскаль и обеспечивающие ей задуманную функциональность:

'Base.pas' – содержит все глобальные переменные программы, основные определения (типы), функции и процедуры,
'ChrOD.pas' – разработка и сохранение эскизов оптических деталей,
'ClcOS.pas' – здесь сосредоточены процедуры и функции по работе с оптикой,
'CtgGls.pas' – модуль для работы с оптическими стёклами,
'CtgRR.pas' – каталог радиусов пробных стёкол,
'Face.pas' – модуль с процедурами, масштабирующими интерфейс программы,
'FitUn.pas' – процедуры для работы с изображениями в FITS-формате,
'GrROS.pas' – процедуры по работе с графикой,
'OptUn.PAS' – процедуры оптимизации,
'PFwrk.pas' – процедуры для работы со зрачковой функцией,
'SvLd.PAS' – модуль с процедурами для загрузки/сохранения данных программы,
'Testing.pas' – процедуры для тестирования ОС,
'TstImg.pas' – процедуры для тестирования ОС по реальным изображениям,
'UTILS.PAS' – сборник вспомогательных процедур,
'WrkOS.PAS' – модуль с основными рабочими процедурами,
'WrkZmx.pas' – процедуры для межпрограммной связи с Земаксом.

2. РАЗРАБОТКА WINDOWS-ПРИЛОЖЕНИЙ В СРЕДЕ DELPHI

Более 10 лет ушло на то, чтобы прочувствовать особенности и освоить программирование Windows-приложений. Нельзя сказать, что в программировании я посторонний: учился в матклассе, с 1963-го приобщен к программированию – сначала на Алгол-60, затем на машинных кодах. И в руководствах по работе в среде Дельфи, вроде, недостатка нет. Однако для полноценной работы всё чего-то не доставало, пока однажды не пришло прозрение.

Процесс разработки оконного приложения состоит из двух разобщённых занятий:

1. Планирования взаимодействия Вашего приложения с компьютерной аппаратурой и с операционной средой и проектирования интерфейса Вашей программы согласно этого взаимодействия.
2. Разработка собственно программного кода, обеспечивающего требуемую функциональность Вашему приложению, которая производится тривиально на языке Паскаль.

Разработка Windows-приложения начинается с разработки его «формы» = «рамы» или главного окна приложения, которая заполняется управляющими элементами:
- кнопками, связывающими программные события с операционной средой Windows,
- редакционными окнами, предоставляющими пользователю возможность изменять рабочие параметры приложения,
- панелями Image, на которые выводится графика программы,
- чек-, комбо- и радио-боксами для выбора вариантов отклика программы,
- метками для текстовых указаний,
и т. д. и т. п.

Однако и этого будет недостаточно. Фирма Microsoft так поставила дело, что полную информацию поставщики компьютерной аппаратуры (клавиатуры, мониторов, мышей, модемов, принтеров, дисков и пр.) предоставляют только ей, а авторы руководств по программированию, соблюдая субординацию, но, в основном, из-за отсутствия этой информации, не вдаются в жизненно важные подробности. Поэтому без помощи программистов-любителей здесь не обойтись. Сведения приходится собирать по крупицам на форумах и в самиздатовских справках, огромную подборку которых собрал Валентин Озеров.

Принципиальную трудность в разработке Windows-приложений составляет вывод графики на панель Image. Тут не нужно вдаваться в нюансы, а просто на примерах, которыми изобилуют все известные руководства программирования в среде Дельфи, приобрести необходимые навыки.

Этот программный код состоит из нескольких обрамляющих строк, внутри которых располагаются гафические операторы языка Паскаль:

```
With Image.Canvas do
Begin
  Rectangle (x0, y0, x1, y1);
  Moveto (x2, y2);
  Lineto (x3, y3);
  Textout (x4, y4, 'Text');
  и т. п.
End;
```

Далее будут рассмотрены основные программые блоки проекта POC, анализируя которые можно составить себе ясное представление о специфике программировния в среде Дельфи. Дистрибутив Delphi-7-Lite выбран по причине его компактности и завершённости.

3. ПРОГОН ЛУЧЕЙ ЧЕРЕЗ ОС

Программа РОС предназначена для расчёта оптических систем типа окуляр или объектив телескопа. Это означает, что в РОС принято, что лучи света приходят к входному зрачку (ВЗ) оптической системы от бесконечно удалённых космических объектов, т.е. параллельными пучками.

Каждый луч, входящий в ОС, преломляясь на каждой её поверхности, завершает свой зигзагообразный путь на фокальной поверхности. Таким образом, основной процедурой расчета является прогон одного луча через всю систему, начиная с входного зрачка и заканчивая фокальной поверхностью.

Целесообразно для каждой системы изначально задать на её ВЗ неизменные координаты входящих лучей, y0 и z0. Варьируя только направляющие косинусы u, v, w параллельного пучка лучей, проходящих через фиксированную координатную сетку на ВЗ, можно исследовать качество ОС по всему её полю зрения. Если при этом для каждого луча одновременно вычислять длину оптического пути, snl, то процедура прогона луча даст три координаты xf, yf и zf на фокальной поверхности и величину волновой ошибки dW, которые в дальнейшем могут быть использованы для оценки качества ОС.

Приняв за оптимальную некоторую длину оптического пути snl0, можно определить величину волновой ошибки OPD (Optical Path Diffrence) для каждого луча, как разницу между реальной и оптимальной длинами путей: dW = snl – snl0.

Совокупность значений волновых ошибок для всего монохроматического пучка лучей, направленного в ОС под одним и тем же углом и привязанного к координатам на входном зрачке, составляет зрачковую функцию ОС, Pupil Function.

Получив после прогона «нулевого» луча координаты xf0, yf0 и zf0 его пересечения с фокальной поверхностью, можно определить поперечную ошибку для каждого последующего луча, как dr = sqrt(sqr(yf – yf0) + sqr(zf – zf0)).

Просуммировав все ошибки для всего пучка лучей, можно определить наибольшую поперечную аберрацию, rmax, и среднеквадратическую ошибку rsq = sqrt(sum(dr) / n), где n – число лучей в пучке. Для волновых ошибок это будут, соответственно, размах ошибки Wmx и dWsq.

Обычно за оптимальную длину snl0 принимают длину оптического пути «нулевого» луча, входящего в ОС через центр ВЗ. Знание координат пересечения нулевого луча с фокальной плоскостью позволяет вычислять все полевые аберрации, в том числе, и дисторсию. В некоторых случаях, желая определить действительный размер пятна в точке поля, предварительно рассчитывают положение «центра тяжести» пятна и все вычисления проводят относительно него. Но нельзя забывать, что при этом пропадает информация о полевых ошибках, особенно о дисторсии. Особенно вредным оказывается такой подход при оптимизации системы, когда оптимизатор начнёт стремиться уменьшить осреднённый по полю и спектру размер пятна, невзирая на наличие дисторсии.

С формульным аппаратом (алгоритмом) прогона лучей через ОС можно ознакомиться в учебниках М.И. Апенко, Б.Н. Бегунова или в монографиях В.Н. Чуриловского, Г.Г. Слюсарева и др. Однако наиболее наглядно и лаконично алгоритмы прогона луча через сферические и планоидные поверхности изложены в книге Г.М. Попова «Асферические поверхности в астрономической оптике» [17].

Последовательность расчёта хода луча через асферические поверхности (АП) второго порядка (АП-2) взята из монографии Г.Г. Слюсарева [6], а через асферические поверхности высоких порядков (АП-ВП) – из книги В.Б. Леоновой «Автоматизация расчётов оптических систем» [12].

Все подпрограммы извлечены из действующего проекта программы РОС.

Тип float определён в модуле base как extended с мантиссой в 10 байт.

Подготовка координатной сетки на входном зрачке.
VzshiftZ и **VzshiftY** – смещение центра ВЗ с оптической оси; используется при расчете децентрированных или внеосевых ОС.

```
// заполнение координатных массивов z0 и y0 в точке m:
procedure setVZdim(a,r:float;var zz,yy:PVector;var m:integer);
begin
 zz^[m]:=r*sin(a)+VZshiftZ;
 yy^[m]:=r*cos(a)+VZshiftY;
 inc(m);
end;

// вычисление расстояния dr между зонами по числу зон n и размеру (Hk – Hv)
// световой зоны и заполнение коодинат для нулевого луча:
procedure set_dr(n:integer;var dr:float;var zz,yy:PVector;var m:integer);
begin
 if vneos then
  dr:=(Hk-VZshift)/n
 else
  dr:=(Hk-Hv)/n;
 yy^[0]:=VZshiftY;
 zz^[0]:=VZshiftZ;
 m:=1;
end;

// расчет сотовой координатной сетки (используется при оптимизации ОС) на ВЗ:
// В модуле base для nZon определено ограничение:
// maxZon=35. Максимальное число лучей: mn = 1 + 3*35*36 = 3781
procedure calcVZ;
var
 i,j,k,m: integer;
 a,c,da,r,dr: Float;
begin
// вычисление зазора между зонами (dr) и заполнение коодинат для нулевого луча
 set_dr(nZon,dr,z0,y0,m);
 for i:=1 to nZon do
 begin
  c:=dr*i;
  k:=6*i;
  da:=pi2/k;
  r:=c;
  if CEkr<>0 then
   r:=c+Hv;
  for j:=0 to k-1 do
  begin
   a:=da*j;
// заполнение координатных массивов z0 и y0 в точке m:
   setVZdim(a,r,z0,y0,m);
  end;
 end;
 mn0:=m;
end;

// 550 лучей распределены равномерно по ВЗ для тестирования ОС
// и вывода информации в графическом виде.
// В модуле base определено:
// NzonMax = 10. Максимальное число лучей: mn = 550, не более.
procedure calcGVZ;
var
 i,j,k,m: integer;
 a,c,da,r,dr: Float;
begin
 set_dr(NzonMax,dr,gz0,gy0,m);
```

```
 for i:=1 to NzonMax do
 begin
  c:=dr*i;
  k:=NzonMax*i;
  da:=pi2/k;
  r:=c;
  if CEkr<>0 then
// для ОС с ЦЭ:
   r:=c+Hv;
  for j:=1 to k do
  begin
   a:=da*j;
   setVZdim(a,r,gz0,gy0,m);
  end;
 end;
 gmn:=m-1;
end;
```

Подпрограмма прогона луча через сферическую поверхность

u,v,w - направляющие косинусы луча, падающего на данную сферу,
x,y,z – исходные координаты пересечения луча с предыдущей поверхностью.
В конце процедуры этими же переменными обозначены направляющие косинусы и координаты пересечения луча, преломлённого на данной поверхности.
PokPre = n – показатель преломления среды, предшествующей данной сфере,
nju = 1/n, rds – радиус кривизны данной сферы.

```
procedure oneWone(var x,y,z,v,w,u:float);
var a,c,d,f,sq: float;
begin
 gbk:=false; // = go back - в случае крушения
 a:=u*x-v*y-w*z;
 c:=2*(x-a*u)-(sqr(a)-sqr(x)-sqr(y)-sqr(z))/rds;
 d:=sqr(u)-c/rds;
 if (d<0) then // признак крушения: нельзя извлечь корень из отрицательного числа
 begin
  gbk:=true;
  exit;
 end;
 f:=sqrt(d);
 a:=a+c/(u+f);
 snl:=snl+pokpre*a; // = длина оптического пути
 // координаты пересечения луча с данной сферой
 x:=a*u-x;
 y:=a*v+y;
 z:=a*w+z;
 sq:=1+sqr(nju)*(d-1);
 if (sq<0) then // признак крушения
 begin
  gbk:=true;
  exit;
 end;
 c:=sqrt(sq)-nju*f;
 a:=-c/rds;
 // направляющие косинусы преломлённого луча
 u:=nju*u+a*x+c;
 v:=nju*v+a*y;
 w:=nju*w+a*z;
end;
```

Подпрограмма прогона луча через асферическую поверхность 2го порядка

exctr – квадрат эксцентриситета данной АП-2.
Остальные обозначения – см. Комментарий к п/п oneWone.

```
procedure oneAP2(var x,y,z,v,w,u:float);
var  a,c,d,b,g,e,sr : Float;
begin
  gbk:=false; // = go back - в случае крушения
  e:=exctr/rds;
  a:=x*u-y*v-z*w;
  g:=a*u-x;
  c:=sqr(x)+sqr(y)+sqr(z)-sqr(a);
  d:=(2+e*g)*g-c/rds;
  sr:=sqr(u)+(d+e*c*sqr(u))/rds;
  if (sr<0) then // признак крушения: ОС не пропустила луч
  begin
   gbk:=true;
   exit;
  end;
  b:=sqrt(sr);
  a:=a-d/(b+u*(1+e*g));
// координаты пересечения луча с данной АП-2:
  x:=a*u-x;
  y:=a*v+y;
  z:=a*w+z;
  snl:=snl+pokpre*a; // = длина оптического пути
  sr:=1+e*(sqr(y)+sqr(z))/rds;
  if (sr<0) then // признак крушения
  begin
   gbk:=true;
   exit;
  end;
  d:=sqrt(sr);
  b:=b/d;
  sr:=1+sqr(nju)*(sqr(b)-1);
  if (sr<0) then // признак крушения
  begin
   gbk:=true;
   exit;
  end;
  g:=(sqrt(sr)-nju*b)/d;
  a:=-g/rds;
// направляющие косинусы преломлённого луча
  u:=nju*u+a*x*(1-exctr)+g;
  v:=nju*v+y*a;
  w:=nju*w+z*a;
end;
```

Подпрограмма прогона луча в области Зейделя через планоидную поверхность пластины Шмидта

Асферичность (стрелка) планоида:

$$x = a20 * \rho^2 + a40 * \rho^4,$$

где $\rho^2 = y^2 / t$, $t = y^2 + z^2$.
Остальные обозначения – см. Комментарии к п/п oneWone.

```
procedure onepSH(var x,y,z,v,w,u:float);
label 1;
var
 i: integer;
```

```
 a,c,dx,g,k,L1,m,n,t,t2,dr : Float;
begin
  gbk:=false;
  L1:=x/u;
// первое приближение для координат точки пересечения с КПШ
  y:=L1*v+y;
  z:=L1*w+z;
  x:=0;
// 6 уточняющих шагов
  for i:=0 to 5 do
  begin
   t:=sqr(y)+sqr(z);
   t2:=sqr(t);
   dx:=t*a20+t2*a40-x;
   k:=2*(a20+t*2*a40); // производная от X
   m:=-k*y;
   n:=-k*z;
   c:=u+v*m+w*n;
   g:=dx/c;
// очередные значения координат
   x:=u*g+x;
   y:=v*g+y;
   z:=w*g+z;
// если изменения координат ничтожны
   if (abs(g)<eps) then
    break;
  end;
  a:=1+sqr(m)+sqr(n);
  snl:=snl+pokpre*(L1+x/u); // оптический путь
  dr:=(1-sqr(nju))*a+sqr(c*nju);
  if (dr<0) then // признак крушения
  begin
   gbk:=true;
   goto 1;
  end;
  a:=(sqrt(dr)-nju*c)/a;
// направляющие косинусы преломлённого луча:
  u:=nju*u+a;
  v:=nju*v+m*a;
  w:=nju*w+n*a;
1: end;
```

Подпрограмма прогона луча через планоидную поверхность высокого порядка

Асферичность (стрелка) планоида ВП:
$x = a20*\rho^2 + a40*\rho^4 + a60*\rho^6 + a80*\rho^8 + a100*\rho^{10} + a120*\rho^{12}$,
где $\rho^2 = y^2 / t$, $t = y^2 + z^2$.
a20 … a120 – коэффициенты высокого порядка.
Остальные обозначения – см. Комментарии к п/п onepSH.

```
procedure onepVP(var x,y,z,v,w,u:float);
label 1;
var
 i: integer;
 a,c,dx,k,L1,m,n,t,t2,dr : Float;
begin
  gbk:=false;
  L1:=x/u;
// первое приближение для координат точки пересечения луча с планоидом ВП
  x:=0;
  y:=L1*v+y;
  z:=L1*w+z;
// 6 шагов приближения
```

```
  for i:=0 to 5 do
  begin
   t:=sqr(y)+sqr(z);
   t2:=sqr(t);
   dx:=t*(a20+t2*(a60+t2*a100))+t2*(a40+t2*(a80+t2*a120))-x;
// производная от dx:
   k:=2*(a20+t*(2*a40+t2*(4*a80+t2*6*a120))+t2*(3*a60+t2*5*a100));
   m:=-k*y;
   n:=-k*z;
   c:=u+v*m+w*n;
   a:=dx/c;
// очередные значения координат
   x:=u*a+x;
   y:=v*a+y;
   z:=w*a+z;
// если изменения координат ничтожны
   if (abs(a)<eps) then
    break;
  end;
  a:=1+sqr(m)+sqr(n);
  snl:=snl+pokpre*(L1+x/u); // оптический путь
  dr:=(1-sqr(nju))*a+sqr(c*nju);
  if (dr<0) then // признак крушения
  begin
   gbk:=true;
   goto 1;
  end;
  a:=(sqrt(dr)-nju*c)/a;
// направляющие косинусы преломлённого луча:
  u:=nju*u+a;
  v:=nju*v+m*a;
  w:=nju*w+n*a;
1: end;
```

Подпрограмма прогона луча через АП-ВП

Величина стрелки x асферической поверхности высокого порядка (АП-ВП) связана с меридиональной y и сагиттальной z высотами на ней следующим соотношением:

$$y^2 + z^2 = 2rx + (e^2 - 1) * x^2 + K3 * x^3 + K4 * x^4 + K5 * x^5 + K6 * x^6,$$

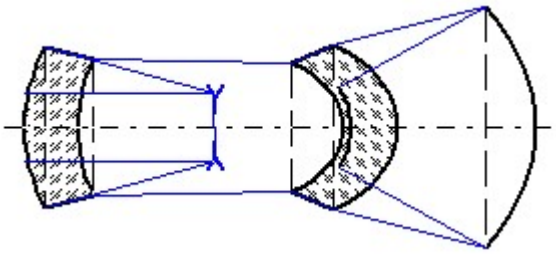
где e^2 – квадрат эксцентриситета поверхности, r – радиус кривизны при вершине АП,
К3 … К6 – коэффициенты высокого порядка, первоначально задаваемые произвольно малыми величинами, а в дальнейшем используемые программой в качестве переменных параметров и уточняемые в процессе оптимизации ОС.
y и z – координаны на поверхности в поперечном сечении;
x – «стрелка», определяемая вдоль оптической оси.
p1 и p2 – 1я и 2я производные, полученные из приведённого соотношения.

```
procedure oneAPVP(var x,y,z,v,w,u:float);
var
 i: integer;
 a,b,f,dx,p1,p2,t1,t2,m,s,nx,ny,nz,q1,q2,g,zn,L,L1,
  y0,z0,x2,y2z: float;
begin
 gbk:=false;
 m:=v/u;
 s:=w/u;
 L1:=x/u;
 y0:=L1*v+y;
 z0:=L1*w+z;
 t2:=sqr(m)+sqr(s)-K2;
 x:=0;
```

```
 y2z:=sqr(y0)+sqr(z0);
 t1:=2*(m*y0+s*z0)-K1;
// 10 шагов приближения:
 for i:=0 to 9 do
 begin
  x2:=sqr(x);
  f:=y2z+x*(t1-x2*(K3+x2*K5))+x2*(t2-x2*(K4+x2*K6));
  p1:=t1+x*(t2-x2*(2*K4+3*x2*K6))*2-x2*(3*K3+5*x2*K5);
  p2:=2*t2-x*(6*K3+x2*20*K5)-x2*(12*K4+x2*30*K6);
  a:=-f/p1;
  b:=-p2/p1;
  dx:=a*(1+a*b);
  x:=x+dx;
// если изменения координат ничтожно малы:
  if (abs(dx)<=eps) then
   break;
 end;
 L:=x/u;
// координаты точки пересечения луча с АП-ВП:
 y:=y0+L*v;
 z:=z0+L*w;
 snl:=snl+pokpre*(L1+L); // оптический путь
// направляющие косинусы нормали к поверхности:
 nx:=-(K1+2*x*(K2+x2*(2*K4+x2*3*K6))+x2*(3*K3+x2*5*K5));
 ny:=2*y;
 nz:=2*z;
 zn:=sqrt(sqr(nx)+sqr(ny)+sqr(nz));
 q1:=(v*ny+w*nz+u*nx)/zn;
 a:=1-sqr(nju)*(1-sqr(q1));
 if (a<=0) then // признак крушения
 begin
  gbk:=true;
  a:=0;
 end;
 q2:=sign(q1)*sqrt(a);
 g:=(q2-nju*q1)/zn;
// направляющие косинусы преломлённого луча
 u:=nju*u+nx*g;
 v:=nju*v+ny*g;
 w:=nju*w+nz*g;
end;
```

Подпрограмма прогона луча через всю ОС

Это – универсальный блок, предназначенный для прогона одного луча через разнородную ОС, имеющую как сферические, так и асферические поверхности, как центрированные, так и смещенные с оси. При этом ОС может быть как обращенная, так и нормально ориентированная в пространстве.



Пример использования подпрограммы прогона лучей через всю ОС для построения её оптической схемы:

В подпрограмме oneAll использованы следующие обозначения:
u,v,w - направляющие косинусы луча, падающего на данную поверхность.

x,y,z – исходные координаты пересечения луча с предыдущей поверхностью; в конце процедуры этими же переменными обозначены направляющие косинусы и координаты луча на фокальной поверхности.
PokPre = n – показатель преломления среды, предшествующей данной поверхности,
nju = 1/n,
rds – радиус кривизны самой поверхности, если это сфера, или привершинной сферы АП.
snl – сумма произведений показателя преломления среды на геометрическую длину хода луча между двумя соседними поверхностями – суммарный оптический путь луча через всю ОС.

```
procedure oneall(var x,y,z,v,w,u:float);
var
 b1,b2,b3: boolean;
 i: integer;
 c,r,kx,ky,kz: float;
begin
 if revOS then
  snl:=0 // начальное значение длины оптического пути для обращённой ОС
 else
  snl:=y*v+z*w; // учет наклона входного зрачка (ВЗ) для оптического пути SNL
// ns – число поверхностей в ОС, включая фокальную
 for i:=0 to ns-1 do
 begin
  rds:=rrs^[i];
  r:=abs(rds);
  exctr:=ex^[i];
  pokPre:=nCurr^[i];
  nju:=eCurr^[i];  // nju = n[i]/n[i+1]
// при расчёте косой ОС:
  if kosina then
   if (r<flat) then
   begin
// сдвигом имитируется наклон сфер на угол "dC" вокруг их осевых вершин
    c:=rds*dltc^[i]; // сдвиг с оси,u-направление сдвига (0..360 gr)
    kx:=c*dltc^[i]/2;
    ky:=c*usc^[i];
    kz:=c*ucc^[i];
    x:=x-kx;
    y:=y-ky;
    z:=z-kz;
   end
   else
   begin
    kx:=0;
    ky:=0;
    kz:=0;
   end;
  x:=ll^[i]-x;
  a30:=a3^[i];              // b:=0; K1 - 1й к-т ВП }
  if r<flat then
   b1:=false
  else
   b1:=true;                //плоскость?
  if (abs(exctr)>eps) then
   b2:=true
  else
   b2:=false;               //АП?
  if (abs(a30)>eps) then
   b3:=true
  else
   b3:=false;               // АП-ВП?
  if ((not b2) and (not b3)) then
   oneWone(x,y,z,v,w,u)     //b:=0; - АП нет
  else
```

```
  if (b2 and (not b3) and (not b1)) then
   oneAP2(x,y,z,v,w,u)      //b:=1; - АП2
  else
  if (b2 and b3 and (not b1)) then // b:=2; - АП-ВП
  begin
   K1:=2*rds;
   K2:=exctr-1;
   K3:=a30;
   K4:=a5^[i];
   K5:=a7^[i];
   K6:=a9^[i];
   oneAPVP(x,y,z,v,w,u);
  end
 else
  if (b1 and b2 and (not b3)) then // b:=3; - КПШ
  begin
   a20:=a2^[i];
   a40:=a4^[i];
   onepSh(x,y,z,v,w,u);
  end
 else
  if (b1 and b2 and b3) then // b:=4; - КПШ-ВП
  begin
   a20:=a2^[i];
   a40:=a4^[i];
   a60:=a6^[i];
   a80:=a8^[i];
   a100:=a10^[i];
   a120:=a12^[i];
   onepVP(x,y,z,v,w,u);
  end;
if kosina then
  begin
   x:=x-kx;
   y:=y+ky;
   z:=z+kz;
  end;
  // возврат на место}
if gbk then
  begin
   inc(outRay);      // число лучей, не прошедших через ОС -
   break;
  end
 else
  inc(nray);         // число лучей, успешно прошедших ОС
// это значение будет использовано при определении коэффициента виньетирования,
// как отношения nRay/mn, где mn число лучей, направленных во ВЗ ОС
 end;
end;
```

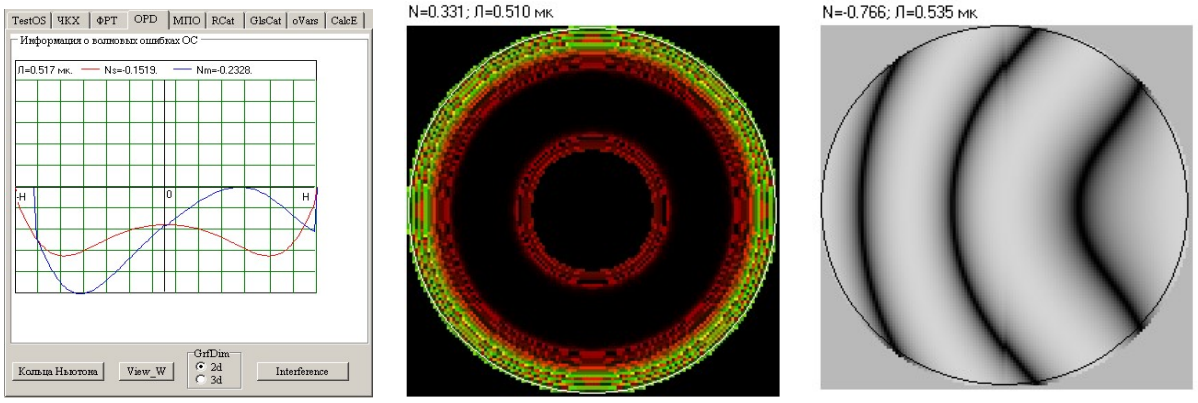
4. РАСЧЕТ ВОЛНОВОЙ ОШИБКИ

На первый взгляд может показаться: подумаешь, кому она нужна, волновая ошибка?! Ведь глаз всё равно реагирует на распределение освещенности, т.е. на размер и расстановку пятен. Однако все критерии качества, такие, как ФРТ и ЧКХ (см. далее), вычисляются посредством анализа волновых ошибок системы, которые составляют зрачковую функцию. Именно поэтому вычисление волновой ошибки – не формальное дополнение, а жизненно необходимая процедура, на которой базируется весь аппарат оценки качества оптики.

Казалось бы, чего проще: одновременно с прогоном лучей через ОС можно подсчитать и длину оптического пути луча. Но не тут-то было: при перефокусировках, необходимых для минимизации оценочных аберраций, приходится фокальную поверхность перемещать не на один микрометр, отчего волновая ошибка катастрофически изменяется на недопустимо большую величину и сильно зависит от кривизны и наклонов фокальной поверхности.

Пытаясь обойти эти затруднения, расчётчики предлагают при вычислении волновой ошибки не доводить прогон луча до фокуса, а выбирать некую «выходную поверхность» – сферу или плоскость – расстояние от которой до фокуса произвольно. При этом упускают из вида, что таким образом останется неизвестным влияние этого расстояния и кривизны «выходной поверхности» на величину волновой ошибки.

Всё выше перечисленное было практически проверено и отвергнуто, поскольку не привело к сколько-нибудь приемлемому и стабильному результату. Поэтому пришлось воспользоваться идеей Г.М. Попова ***рассчитывать волновую ошибку для обращённой ОС***, когда прогон лучей производится в обратном направлении, от фокальной поверхности в сторону ВЗ. Именно только таким образом удалось окончательно закрыть проблему точного расчёта зрачковой функции.



Слева – волновая ошибка системы в меридиональном и сагаттальном сечениях зрачковой функции; посредине – интерференция в кольцах Ньютона, справа – в полосах.

Подпрограмма оптимизации последнего отрезка

Как было уже сказано, расчёт волновой ошибки производится для обращённой ОС, когда исследуемая ОС развёрнута задом-неперёд. Основной задачей процедуры оптимизации последнего отрезка является его минимизация. Оптимизацией последнего отрезка достигается не только минимально возможный размах волновой ошибки, что важно для объективной оценки качества системы, но становится возможным учёт формы волнового фронта и его наклонов.

Подпрограмма производит поиск такого последнего отрезка Sp (для нормально расположенной, не обращённой, ОС), при котором оптические пути нулевого и краевого лучей ока-

зываются равными между собой, а волновая ошибка для зоны 0.707 – минимальной. В итоге одновременно находится новый центр для ВЗ и определяются вспомогательные величины op1 – op4 , используемые для коррекции наклона волнового фронта в подпрограмме расчета зрачковой функции CalcPFv, что гарантирует минимальные характеристики для волновой ошибки системы, а, значит, наивысшие значения для критериев качества ОС.

Здесь: Xfs – стрелка на фокальной сфере на высоте Yf над оптической осью, откуда исходит луч, а ui, vi и wi – направляющие косинусы исходящего луча, определяемые геометрически через фокус ОС и вычисляемые в подпрограмме CalcPFv.

```
procedure OptSpBkOS(yf : float);
var
  p1,p2,v2,w2,u2,x,y,z,v,w,u,xfs: float;
```

Вспомогательная функция для определения длины оптического пути в меридиональной плоскости:

```
function MB1:float;
begin
 x:=xfs;
 y:=yf;
 z:=0;
 v:=vi;
 w:=0;
 u:=ui;
 oneAll(x,y,z,v,w,u);
 result:=SNL;
end;
```

Вспомогательная функция для вычисления волновой ошибки косого луча:

```
function MB2:float;
begin
 x:=xfs;
 y:=yf;
 z:=0;
 v:=v2;
 w:=w2;
 u:=u2;
 oneAll(x,y,z,v,w,u);
 result:=SNL-(y-yc)*vi-(z-zc)*wi-optPath0;
end;

// основной блок:
var
 k: integer;
 c: float;
begin
 if (Rfs < flat) then
  xfs:=Rfs-sign(Rfs)*sqrt(sqr(Rfs)-sqr(yf))
 else
  xfs:=0;
// первоначальное смещение фокуса:
 c:=0.256;
 p1:=MB1;
 ll^[0]:=ll^[0]+c;
 p2:=MB1;
 for k:=0 to 4 do
 begin // optPath0=>min
  if p2<p1 then
  repeat
   p1:=p2;
   ll^[0]:=ll^[0]+c;
   p2:=MB1;
```

```
        until p2>p1
        else
        repeat
         p1:=p2;
         ll^[0]:=ll^[0]-c;
         p2:=MB1;
        until p2<p1;
      // уменьшаем смещение и 5 раз повторяем этот цикл
        c:=c/8;
       end;
       optPath0:=MB1;
      // новый центр ВЗ:
       xc:=x;
       yc:=y;
       zc:=z;
       SpBkOS:=ll^[0]; // Sp установлено с точностью 1 / 2^15 ~ 0.03 мкм
       v2:=(H0-yf)/Fl; // верхний край ВЗ
       w2:=0;
       u2:=sqrt(1-sqr(v2));
       op2:=MB2;
       v2:=-(H0+yf)/Fl; // нижний край ВЗ
       w2:=0;
       u2:=sqrt(1-sqr(v2));
       op3:=MB2;
       v2:=vi;
       w2:=H0/Fl;  // край ВЗ в сагиттальной плоскости
       u2:=sqrt(1-sqr(v2)-sqr(w2));
       op4:=MB2;
      end;
```

Подпрограмма вычисления волновой ошибки для обращённой ОС

Вычисление зрачковой функции обращённой ОС в долях длины волны Lam в меридиональном сечении для точки на фокальной поверхности на расстоянии Yf от центра поля зрения. Путём поворота центрированной системы относительно оптической оси любой косой луч всегда можно расположить в меридиональной плоскости, поэтому волновую ошибку системы всегда считают для полевой точки, находящейся в меридиональной плоскости.
q – число сечений ВЗ в меридиональном и сагиттальном направлениях,
ww – массив значений волновой ошибки на ВЗ (зрачковая функция ОС).

```
function CalcPFv(var ww:PVector;yf,Lam:float;q:integer):boolean;
var
 Apodiz,zonAberAdd: boolean;
 i,j,k,n,q2,q4,qrL,qrR,k0: integer;
 a,b,ba,c0,c2,c4,da,Hv7,ka,Lad,mn,mx,r2,s,iy,jz,y1,y2,x,y,z,v,w,u,xfs: float;
begin
 revOS:=true;
 //разворачиваем ОС задом-наперёд:
 setCurrOS;
 q2:=q div 2;
 q4:=q2 div 2;
 yf:=yf*mirAng; // учёт разворота пакетов для мир
 Lad:=1e3/Lam;  // переводим микрометры в миллиметры
 k0:=q4*q4;
 if yf=0 then
  yf:=eps;
 if abs(Rfs)<flat then // вычисляем стрелку на фокальной сфере
  xfs:=Rfs-sign(Rfs)*sqrt(sqr(Rfs)-sqr(yf))
 else
  xfs:=0;
 da:=H0*difrImMast/q4; // шаг по ВЗ
```

```
 if Rastajki then      // если нужно учесть затенение ВЗ растяжками
 begin
  qrL:=q2-round(trast/2/da);
  qrR:=q2+round(trast/2/da);
 end
 else
 begin
  qrL:=q2;
  qrR:=q2;
 end;
 ka:=pi*zonAberFreq/H0;
// если нужно учесть дополнительную зональную волновую ошибку
 if zonAberAmp<>0 then
  zonAberAdd:=true
 else
  zonAberAdd:=false;
// направляющие косинусы нулевого луча, идущего от фокуса к ВЗ:
 vi:=yf/Fl;
 wi:=0;
 ui:=sqrt(1-sqr(vi));
// оптимизируем последний отрезок:
 OptSpBkOS(yf);
 c2:=(op2-op3)/2/sqrt(H2); // наклон волнового фронта
// для вычисления стрелки на ближайшей сфере сравнения (БСС) к волновому фронту:
 c4:=op4/H2;
 mx:=0;
 mn:=0;
// если ОС аподизирована на нуль волновой ошибки:
 if frmMain.Apodizat_ChBox.State=cbChecked then
  Apodiz:=true
 else
  Apodiz:=false;
 // далее обходится весь круглый ВЗ - от «-q4» до «+q4»;
 // остальные точки в зрачковой функции - нулевые, они составляют «фальшборт»,
 // необходимый процедуре БПФ для корректного анализа центральной, рабочей, зоны
 // зрачковой функции.
 for i:=q4 to q2+q4-1 do
 if (i<=qrL) or (i>=qrR) then
 begin
  if i<=qrL then
   iy:=da*(qrL-i-0.5);
  if i>=qrR then
   iy:=da*(qrR-i+0.5);
  y1:=iy;
  y2:=sqr(y1);
  for j:=q4 to q2+q4-1 do
  if (j<=qrL) or (j>=qrR) then
  begin
   if j<=qrL then
    jz:=da*(qrL-j-0.5);
   if j>=qrR then
    jz:=da*(qrR-j+0.5);
   r2:=y2+sqr(jz);
// если точка входит во ВЗ, экранированный до Hv, то из фокальной точки
// с координатами xfs, yfs и zfs в сторону ВЗ посылается луч, имеющий
// направляющие косинусы v, w, u :
   if (r2<=H2) and (r2>=Hv2) then
   begin
    x:=xfs;
    y:=yf;
    z:=0;
    v:=(iy-yf)/Fl;
    w:=jz/Fl;
    u:=sqrt(1-sqr(v)-sqr(w));
    oneAll(x,y,z,v,w,u);
```

```
// вычитаем наклоны ВЗ и стрелку на сфере, ближайшей к волновому фронту:
     b:=SNL-optPath0-(y-yc)*vi-(z-zc)*wi-c2*y1-c4*r2;
     if zonAberAdd then
      b:=b+zonAberAmp*sin(ka*sqrt(r2));
     // dW/Lam:
     b:=b*Lad;
// ищем максимум и минимум у зрачковой функции:
     if abs(b)>abs(mx) then
      mx:=b;
     if abs(b)<abs(mn) then
      mn:=b;
     ba:=b;
    ww^[i*q+j]:=ba;
   end;
  end;
 end;
 //с учётом знака:
 maxW:=mx-mn;
 mx:=0;
 mn:=0;
 s:=0;
 k:=0;
 for i:=q4 to q2+q4-1 do
 if (i<=qrL) or (i>=qrR) then
 begin
  if i<=qrL then
   iy:=da*(qrL-i-0.5);
  if i>=qrR then
   iy:=da*(qrR-i+0.5);
   y2:=sqr(iy);
  for j:=q4 to q2+q4-1 do
  if (j<=qrL) or (j>=qrR) then
  begin
   if j<=qrL then
    jz:=da*(qrL-j-0.5);
   if j>=qrR then
    jz:=da*(qrR-j+0.5);
   r2:=y2+sqr(jz);
   if (r2<=H2) and (r2>=Hv2) then
   begin
    n:=i*q+j;
    a:=ww^[n];
    // вычитаем "пьедестал"
    if abs(a)<>0 then
     b:=a-maxW
    else
     b:=a;
    if Apodiz then
      b:=0;
    ww^[n]:=b;
    if abs(b)>abs(mx) then
     mx:=b;
    if abs(b)<abs(mn) then
      mn:=b;
    s:=s+sqr(b);
    inc(k);
   end;
  end;
 end;
 maxW:=mx-mn;
 wsqv:=sqrt(s/k);
 if maxW=0 then
  if mx=0 then
   maxW:=1
  else
```

```
   maxW:=mx;
 CalcPFv:=true;
// возвращаемся к нормально ориентированной ОС:
 revOS:=false;
setCurrOS;
end;
```

5. УЧЁТ ДИФРАКЦИИ ПОСРЕДСТВОМ БПФ

Те, кому приходилось вычислять определённые интегралы, знают, что, во-первых, наивысшую точность даёт квадратура Гаусса, а, во-вторых, даже Гаусс не в силах вычислить точное значение определённого интеграла. Та же неопределённость препятствует исследованиям дифракционных явлений: прогон десятков тысяч лучей не даёт уверенного результата.

Поэтому появление математического аппарата под названием «преобразование Фурье» в корне изменило ситуацию. Доступной в инженерных вычислениях процедура стала только после открытия быстрых алгоритмов этих преобразований (БПФ).

В связи с указанными трудностями возникает резонное сомнение: для чего же тогда учитывать дифракцию и что это реально даёт?!

Исследования показывают, что размер пятна рассеяния, полученный путём геометрического прогона лучей, не соответствует действительности: дифракция не просто структурирует пятно, преобразуя равномерную интенсивность внутри него в волнообразную, но в принципе изменяет распределение яркости по пятну, ***обесценивая таким образом результаты лучевых расчетов пятен рассеивания***.

Приведённый ниже программный код обработки зрачковой функции процедурой БПФ также принадлежит перу французского математика-программиста Jean Debord. Его использование оказалось целесообразнее прикрепления к программе POC известной динамической библиотеки FFTW3.DLL, занимающей более полумегабайта дискового пространства и, соответсвенно, значительный объём оперативной памяти компьютера.

```
procedure doFFT(var InArray: PCompVec; NumSamples: Integer);

function IsPowerOfTwo(X:Integer):Boolean;
var I,Y: Integer;
begin
 Y:=2;
 for I:=1 to 25 do
 begin
  if X=Y then
  begin
   IsPowerOfTwo:=True;
   Exit;
  end;
  Y:=Y shl 1;
 end;
 IsPowerOfTwo := False;
end;

function NumberOfBitsNeeded(Pof2:Integer):Integer;
// вычисляет показатель двойки для числа строк/столбцов
var i: Integer;
begin
// 2^26 = максимальное целое для IBM = 69 108 864 = 8192x8192
 for i:=0 to 26 do
  if ((Pof2 and (1 shl i)) <> 0) then
 begin
  NumberOfBitsNeeded:=i;
```

```
  Exit;
 end;
end;

function ReverseBits(ix,NBits:Integer):Integer;
// обращение номеров элементов
var i,Rev: Integer;
begin
 Rev:=0;
 for i:=0 to NBits-1 do
 begin
  Rev:=(Rev shl 1) or (ix and 1);
  ix:=ix shr 1;
 end;
 ReverseBits:=Rev;
end;

var
 NumBits,I,J,K,N,BlockSize,BlockEnd: Integer;
 Delta_angle,Delta_ar,Alpha,Beta,Tr,Ti,Ar,Ai: Float;
 outArray: PCompVec;
begin
// проверяем, является ли число членов массива производным 2
 if not IsPowerOfTwo(NumSamples) or (NumSamples < 2) then
 begin
  showMessage('NSec='+inttostr(NSec)+
   '- д.б. степенью 2:'+vk+'32,64,128,...');
  Exit;
 end;
// определяем показатель степени с основанием 2
 NumBits := NumberOfBitsNeeded(NumSamples);
// формируем рабочие массивы
 DimCompVec(outArray,NumSamples);
 for I := 0 to NumSamples - 1 do
 begin
  J := ReverseBits(I, NumBits);
  OutArray^[J].re := InArray^[I].re;
  OutArray^[J].im := InArray^[I].im;
 end;
 BlockEnd := 1;
 BlockSize := 2;
// Преобразуем входной массив
 while BlockSize <= NumSamples do
 begin
  Delta_angle := pi2 / BlockSize;
  Alpha := Sin(0.5 * Delta_angle);
  Alpha := 2.0 * Alpha * Alpha;
  Beta := Sin(Delta_angle);
  I := 0;
  while I < NumSamples do
  begin
   Ar:=1;
   Ai:=0;
   J := I;
```

```
   for N := 0 to BlockEnd - 1 do
   begin
    K := J + BlockEnd;
    Tr := Ar * OutArray^[K].re - Ai * OutArray^[K].im;
    Ti := Ar * OutArray^[K].im + Ai * OutArray^[K].re;
    OutArray^[K].re := OutArray^[J].re - Tr;
    OutArray^[K].im := OutArray^[J].im - Ti;
    OutArray^[J].re := OutArray^[J].re + Tr;
    OutArray^[J].im := OutArray^[J].im + Ti;
    Delta_ar := Alpha * Ar + Beta * Ai;
    Ai := Ai - (Alpha * Ai - Beta * Ar);
    Ar := Ar - Delta_ar;
    Inc(J);
   end;
   I := I + BlockSize;
  end;
  BlockEnd := BlockSize;
  BlockSize := BlockSize shl 1;
 end;
// заменяем входные данные выходными
 for i:=0 to NumSamples-1 do
 begin
  inArray^[i].re:=outArray^[i].re;
  inArray^[i].im:=outArray^[i].im;
 end;
// освобождаем память
 DelCompVec(outArray,NumSamples);
end;
```

6. РАСЧЁТ ФРТ И ЧКХ

Имея в своём распоряжении зрачковую функцию, легко получить дифракционное пятно рассеяния, ФРТ, и частотно-контрастную характеристику, ЧКХ.

Процедура **preparePF** производит преобразование фазы волновой ошибки в sin-cos значения, и она используется во всех процедурах перед выводом информации о форме волнового фронта и пр.

Нет необходимости подвергать преобразованию весь двумерный массив зрачковой функции – достаточно обработать центральный круг: если размерность всего массива q*q, то информация об ошибке ОС содержится в круге диаметром q2 = q/2. При этом q4 = q2/2 - полудиаметр круга.

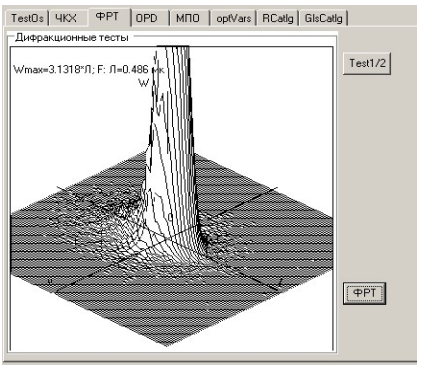
Величина da = H0/q4 – линейный масштаб по ВЗ.

В подпрограмме учитывается толщина растяжек **trast**, на которых держится вторичное или диагональное зеркало телескопа.

```
// befor CalcPSF
procedure preparePF(q:integer;var wc,ws:PVector);
var
 b1: boolean;
 i,j,n,q2,q4,qrL,qrR: integer;
 d3,da,iy,jz,r2,y2: float;
begin
 q2:=q div 2;
 q4:=q2 div 2;
 da:=H0*difrImMast/q4;
 if Rastajki then
 begin
  qrL:=q2-round(trast/2/da);
  qrR:=q2+round(trast/2/da);
 end
 else
 begin
  qrL:=q2;
  qrR:=q2;
 end;
// сканируем весь ВЗ, от –H до +H; 0.5 – смещение на полделения.:
 for i:=q4 to q2+q4-1 do
 if (i<=qrL) or (i>=qrL) then
 begin
  if i<=qrL then
   iy:=da*(q2-i-0.5);
  if i>=qrR then
   iy:=da*(qrR-i+0.5);
// Для внеосевой ОС учитываем смещение ВЗ:
  if vneos then
   iy:=iy+VZshiftY;
  y2:=sqr(iy);
  for j:=q4 to q2+q4-1 do
  if (j<=qrL) or (j>=qrR) then
```

```
  begin
   if j<=qrL then
    jz:=da*(qrL-j-0.5);
   if j>=qrR then
    jz:=da*(qrR-j+0.5);
   if vneos then
    jz:=jz+VZshiftZ;
   r2:=y2+sqr(jz);
   b1:=false;
```
// Учитываем (**b1:=true**) только рабочую (кольцевую) область ВЗ:
```
   if (r2<=H2) then
    b1:=true;
```
// Исключаем (**b1:=false**) экранированную область ВЗ:
```
   if ((not vneos) and (r2<Hv2)) then
    b1:=false;
   if b1 then
   begin
    n:=j+i*q;
```
// волновая ошибка **ws** определена в долях длины волны, поэтому **d3** – это фаза:
```
    d3:=pi2*ws^[n];
    wc^[n]:=cos(d3);
    ws^[n]:=sin(d3);
   end;
  end;
 end;
end;
```

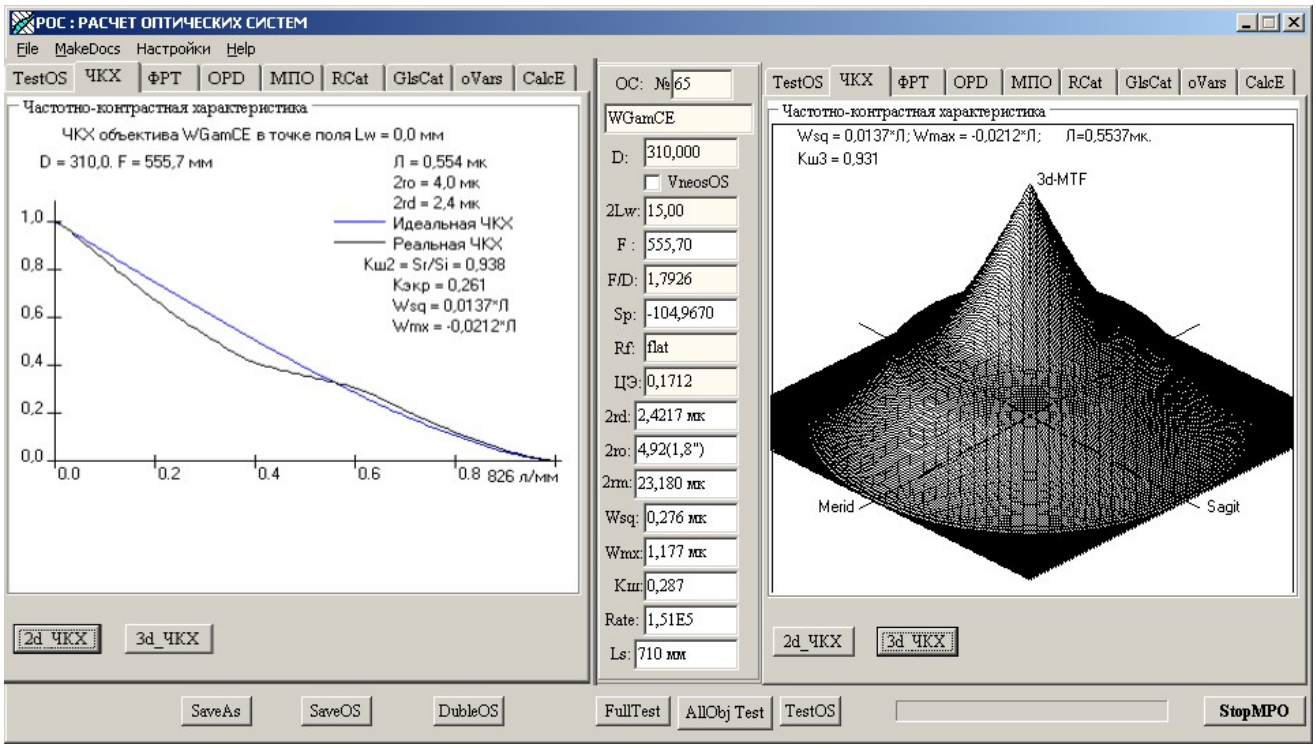
Псевдо-объёмный График ФРТ:



// транспонирование квадрантов большой матрицы,
// образованной из одномерного массива V
```
procedure ReorgVector(var V:PVector;q:integer);
var
 i,j,k,q2: integer;
 ww,ff: PMatrix;
begin
 q2:=q div 2;
// формируем два вспомогательных массива:
 DimMatrix(ww,q2,q2);
 DimMatrix(ff,q2,q2);
 for i:=0 to q2-1 do
  for j:=0 to q2-1 do
// … и вписываем в них данные:
  begin
   ww^[i]^[j]:=V^[q*i+j];
   ff^[i]^[j]:=V^[q*(q2+i)+q2+j];
  end;
// квадранты 1 и 4
 for i:=0 to q2-1 do
  for j:=0 to q2-1 do
  begin
   V^[q*i+j]:=ff^[i]^[j];
   V^[q*(q2+i)+q2+j]:=ww^[i]^[j];
```

```
  end;
 for i:=0 to q2-1 do
  for j:=0 to q2-1 do
  begin
   ww^[i]^[j]:=V^[q*i+q2+j];
   ff^[i]^[j]:=V^[q*(q2+i)+j];
  end;
// квадранты 2 и 3
 for i:=0 to q2-1 do
  for j:=0 to q2-1 do
  begin
   V^[q*i+q2+j]:=ff^[i]^[j];
   V^[q*(q2+i)+j]:=ww^[i]^[j];
  end;
// опускаем левую половину результирующей матрицы на 1 строку:
 for i:=q-1 downto 1 do
  for j:=0 to q2-1 do
  begin
   k:=q*i+j;
   V^[k]:=V^[k-q];
  end;
//
 DelMatrix(ww,q2,q2);
 DelMatrix(ff,q2,q2);
end;


// CalcPSFarr производит БПФ входящей зрачковой
// функции (ФЗ), вычисляет в каждой точке яркость как сумму квадратов мнимой и реальной
// составляющих и реорганизует массив, переставляя местами квадранты результирующей
// матрицы, развёрнутой, как и все входящие массивы, в одномерный массив-вектор
function CalcPSFarr(
  wc: PVector;      // массив ФЗ на q*q
  m0: integer;      // = q
  var ws: PVector; // выходной массив Pupil Spread Function
  var mnx: float   // максимальное значение в массиве данных
  ): boolean;       // false==out
var
 a,mx: float;
 i,k0,q: integer;
 outdat: PCompVec;
begin
 q:=m0;
 k0:=q*q;
// формируем комплексный выходной рабочий массив
 DimCompVec(outdat,k0);
 for i:=0 to k0-1 do
 begin
  outdat^[i].re:=wc^[i];
  outdat^[i].im:=ws^[i];
 end;
// … и отправляем его на БПФ:
 doFFT(outdat,k0);
 mx:=0;
```

```
// возводим реальную и мнимую результирующие в квадрат и суммируем:
 for i:=0 to k0-1 do
 begin
  a:=sqr(outdat^[i].re)+sqr(outdat^[i].im);
  ws^[i]:=a;
// ищем в массиве максимальное значение:
  if (a > mx) then
   mx:=a;
 end;
// исключаем деление на нуль при нормировании массива:
 if (mx = 0) then
  mx:=1;
 mnx:=mx;
// освобождаем память от рабочего массива:
 DelCompVec(outdat,k0);
// реорганизуем результат, переставляя у матрицы квадранты:
 ReorgVector(ws,q);
// сообщаем программе об успешном завершении работы:
 CalcPSFarr:=true;
end;
```



2х- и 3х-мерные графики ЧКХ

```
// вычисление массива VV данных для ЧКХ
function CalcMTFarr(
 var vv:PVector;// MTF q-array
 q0:integer;    // q==2*D
 L:float        // =Lw
 ):boolean;     // false = out
var
 i,j,k0,n,q,q2: integer;
 a,mx: float;
 wc,ws: PVector;
 outdat: PCompVec;
```

```
begin
 q:=q0;
 k0:=q*q;
 q2:=q div 2;
// два массива: реальная и мнимая составляющие
 DimVector(wc,k0);
 DimVector(ws,k0);
 if not CalcPFv(ws,L,Lambda,q) then
// если зрачковая функция не посчитана
 begin
  CalcMTFarr:=false;
  exit;
 end;
// подготавливаем массивы данных:
 preparePF(q,wc,ws);
 if not CalcPSFarr(wc,q,ws,mx) then
// если массив данных ФРТ не посчитан:
 begin
  CalcMTFarr:=false;
  exit;
 end;
 // in ws - PSF
 DimCompVec(outdat,k0);
// готовим массивы для второго БПФ:
 for i:=0 to k0-1 do
 begin
  outdat^[i].re:=0;
  outdat^[i].im:=ws^[i];
 end;
// отправляем массивы на БПФ:
 doFFT(outdat,k0);
// производим обработку результата БПФ:
 for i:=0 to q-1 do
  for j:=0 to q-1 do
  begin
   n:=j+i*q;
   wc^[n]:=sqrt(sqr(outdat^[n].re)+sqr(outdat^[n].im));
 end;
// освобождаем память:
 DelCompVec(outdat,k0);
// переставляем квадранты результирующей матрицы:
 ReorgVector(wc,q);
 n:=q*(q2-1)+q2;
 mx:=0;
 for i:=0 to q2-1 do
// переносим данные из массива WC в массив VV и ищем максимум:
 begin
  a:=wc^[n+i];
  vv^[i]:=a;
  if abs(a)>mx then
   mx:=abs(a);
 end;
 mxm:=mx;
 if mxm=0 then
```

```
// предотвращаем деление на нуль:
  mxm:=1;
 for i:=0 to q2-1 do
// нормируем массив VV
  vv^[i]:=vv^[i]/mxm;
// освобождаем память:
 DelVector(wc,k0);
 DelVector(ws,k0);
// сообщаем об успехе:
 CalcMTFarr:=true;
end;

// расчёт данных для построения 3х-мерной ЧКХ
procedure Calc3dMTF(var ps,mt:PVector;var mx:float);
var
 i,j,k0,n,q: integer;
 a,L: float;
 wc,ws: PVector;
 outdat: PCompVec;
begin
 q:=NSec;
 k0:=q*q;
// формируем вспомогательные массивы
 DimVector(wc,k0);
 DimVector(ws,k0);
 for n:=0 to k0-1 do
// переносим зрачковую функцию во вспомогательный массив
  ws^[n]:=puplFunc^[n];
// делаем предобработку ФЗ
 preparePF(q,wc,ws);
 if CalcPSFarr(wc,q,ws,mx) then
// если массив ФРТ получен успешно, работаем дальше
 Begin
// формируем массив комплексных чисел и заполняем его
  DimCompVec(outdat,k0);
  for i:=0 to k0-1 do
  begin
   outdat^[i].re:=0;
   outdat^[i].im:=ws^[i];
  end;
// производим БПФ
  doFFT(outdat,k0);
  mx:=0;
  for i:=0 to q-1 do
   for j:=0 to q-1 do
// обрабатываем полученные данные
   begin
    n:=j+i*q;
    a:=sqrt(sqr(outdat^[n].re)+sqr(outdat^[n].im));
    mt^[n]:=a;
    if a>mx then
// находим максимальное значение
     mx:=a;
   end;
```

```
// освобождаем память
   DelCompVec(outdat,k0);
// преобразуем результирующую матрицу
   ReorgVector(mt,q);
 end;
// освобождаем память
 DelVector(wc,k0);
 DelVector(ws,k0);
end;
```

7. ТЕСТИРОВАНИЕ ОС ПО ИСКУССТВЕННЫМ ОБЪЕКТАМ И СНИМКАМ РЕАЛЬНЫХ ОБЪЕКТОВ

При обсуждении на Астрофоруме программы РОС Андреем Зубаревым было высказано пожелание производить программное тестирование оптических систем по искусственным объектам: по одной или двум звёздам, по двум линиям или многоштриховым мИрам. Игорь Николаевич Панкратов посоветовал попробовать тестировать по снимку реального небесного объекта, например, по снимку Юпитера.

Оба совета были учтены в программном коде РОС в процедуре **MakeImgPF**.

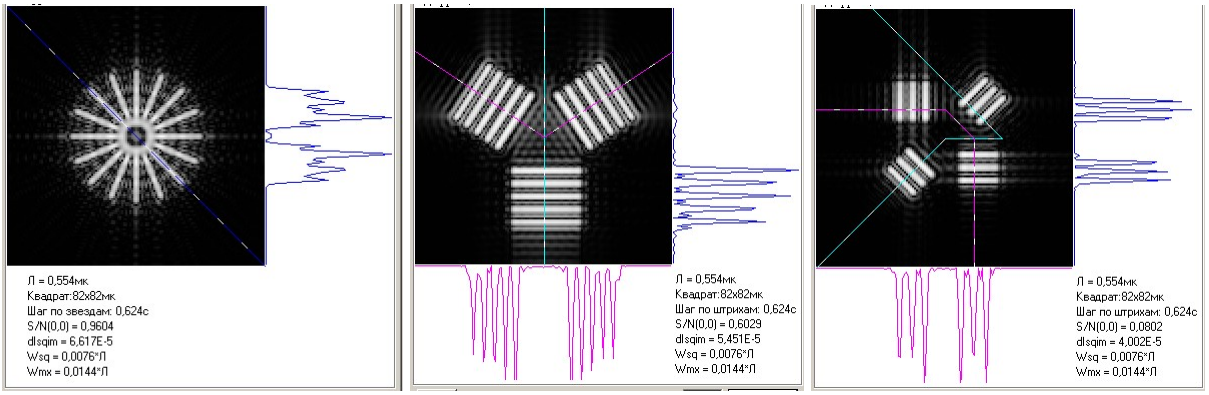
В этом программном блоке синтезированная ФЗР формируется из одиночной ФЗ1 по следующему алгоритму:

1. Тест-объект разбивается на ряд точек, угловое расстояние между которыми выбирается меньше разрешающей способности объектива, так, чтобы после синтеза эти точки сливались в сплошную линию;
2. Для каждой точки тест-объекта вычисляются углы смещения в меридиональной и сагиттальной плоскостях относительно его центральной точки:
3. В процедуре **makeArr** производится синтез результирующей ФЗР из ФЗ1 путем наклона последней на углы, посчитанные в п. 2.

Опыт показал: поскольку угловые размеры тест-объектов составляют малую величину, то нет необходимости вычислять ФЗ для каждой точки тест-объекта, а достаточно иметь ФЗ1 для его центральной точки, соответствующей углу поля, для которого обычно уже посчитана ФЗ1. Это экономит время тестирования и практически не вносит заметных ошибок.

```
procedure MakeImgPF(m:byte;var ww1,ww2:PVector);
// pupil Func for: 2 or 4stars, 3 or 4-Mir
var
 q,q2,q4,qrL,qrR,m0: integer;
 da,dc: float;

procedure makeArr(k0:integer;v0,w0,db,de:float);
// формирование синтезированной ФЗР по ФЗ1 одной звезды
var
 b1: boolean;
 i,j,k,n: integer;
 b,c,s1,s2,iy,jz,r2,y2: float;
begin
 for i:=q4 to q4+q2-1 do
// поиск на ФЗ незатенённой рабочей точки
 if (i<=qrL) or (i>=qrR) then
 begin
  if i<=qrL then
   iy:=da*(qrL-i-0.5);
  if i>=qrR then
   iy:=da*(qrR-i+0.5);
  if vneos then
   iy:=iy+VZshiftY;
  y2:=sqr(iy);
```

```
  for j:=q4 to q4+q2-1 do
  if (j<=qrL) or (j>=qrR) then
  begin
   if j<=qrL then
    jz:=da*(qrL-j-0.5);
   if j>=qrR then
    jz:=da*(qrR-j+0.5);
   if vneos then
    jz:=jz+VZshiftZ;
   r2:=sqr(jz)+y2;
   b1:=false;
   if (r2<=H2) then
    b1:=true;
   if ((not vneos) and (r2<sqr(Hv))) then
    b1:=false;
   if b1 then
// если найденная точка принадлежит незатенённой области ВЗ
   begin
    n:=j+i*q;
    if band then
// если необходимо посчитать интерференционную полосу
     PuplFunc^[n]:=iy*v0+ PuplFunc ^[n]
   Else
// иначе синтезируем ФЗР
    begin
     s1:=0;
     s2:=0;
     c:=PuplFunc^[n];
     for k:=0 to k0-1 do
// синтезируем линию из К-точек
     begin
      b:=(iy*(v0+db*k)+jz*(w0+de*k)+c)*pi2;
// суммируем фазы
      s1:=s1+cos(b);
      s2:=s2+sin(b);
     end;
// охраняем результат в рабочих массивах
     ww1^[n]:=ww1^[n]+s1;
     ww2^[n]:=ww2^[n]+s2;
    end;
   end;
  end;
 end;
end;
```



Тестирование ОС по синтезированным мирам: радиальной, 3х- и 4х-штриховой.

```
procedure Img;
// преобразование изображения реального тест-объекта в ФЗ
procedure makeIm(m:float;v0,w0:float);
var
 i,j: integer;
 iy,jz: float;
procedure summator(n:integer;ka:float);
var
 b: float;
begin
 b:=(iy*v0+jz*w0+PuplFunc^[n])*pi2;
 ww1^[n]:=cos(b)*ka+ww1^[n];
 ww2^[n]:=sin(b)*ka+ww2^[n];
end;

var
 k: integer;
 a,r2,y2: float;
begin
 for i:=q4 to q4+q2-1 do
 if (i<=qrL) or (i>=qrR) then
 begin
  if i<=qrL then
   iy:=da*(qrL-i-0.5);
  if i>=qrR then
   iy:=da*(qrR-i+0.5);
  y2:=sqr(iy);
  for j:=q4 to q4+q2-1 do
  if (j<=qrL) or (j>=qrR) then
  begin
   if j<=qrL then
    jz:=da*(qrL-j-0.5);
   if j>=qrR then
    jz:=da*(qrR-j+0.5);
   r2:=y2+sqr(jz);
   if (r2<=H2)
   then
    summator(q*i+j,m);
  end;
 end;
end;
```

```
var
 i,j,i0,j0,h,w: integer;
 di,dy,Isa,Isq,mx: float;
 PrgsBar: TProgressBar;
begin
 w:=fitImage.col;
 h:=fitImage.lin;
 Isq:=fitImage.Isqv;
 i:=nSec div 2;
 if w>i then
  w:=i;
 if h>i then
  h:=i;
 PrgsBar:=frmMain.PAT_PrgsBar;
 PrgsBar.position:=0;
 i0:=w div 2;
 j0:=i0;
 PrgsBar.Max:=h;
 PrgsBar.Step:=1;
 for j:=0 to h-1 do
 begin
  dy:=(j-j0)*dc;
// устраняем «зависание» компьютера
  Application.ProcessMessages;
  for i:=0 to w-1 do
  // на вход БПФ будет подан не сигнал, а его контраст!
   makeIm(kadr^[j]^[i]/Isq,(i-i0)*dc,dy);
// прогресс-бар добавляет «кубик» в информационную линейку
  PrgsBar.StepIt;
 end;
end;

// 7*7 Stars
procedure PolSta;
var
 ot: PVector;

 procedure BuildPFforPolSta(k,n:integer);
// преобразование меридиональной ФЗ в пространственную ФЗ:
 var
  b: boolean;
  i,j: integer;
  z: float;
 begin
  if (k = 0) and (n = 0) then
   exit;
  z:=(abs(k)+abs(n))/sqrt2;
  b:=false;
  if (k*n > 0) then
   b:=true;
// зрачковая функция в массиве ot повёрнута на 90 градусов
  for i:=q4 to q4+q2-1 do
   for j:=q4 to q4+q2-1 do
    ot^[q*j+i]:=PuplFunc^[q*i+j];
```

```
// модифицируем зрачковую функцию:
   if (k = 0) then
// для горизонтальной оси - поворот зрачковой функции на 90 градусов
   for i:=q4 to q4+q2-1 do
    for j:=q4 to q4+q2-1 do
     begin
      PuplFunc^[q*i+j]:=ot^[q*i+j];
     end
   else
   if b then
// если знаки у k и n одинаковые (квадранты 1 и 3):
   for i:=q4 to q4+q2-1 do
    for j:=q4 to q4+q2-1 do
     begin

PuplFunc^[q*i+j]:=(abs(n)*ot^[q*i+j]+abs(k)*PuplFunc^[q*i+j])/z;
     end
  else
  if (n < 0) then
// если знаки разные и n отрицательный (2 квадрант):
   for i:=q4 to q4+q2-1 do
    for j:=q4 to q4+q2-1 do
     begin
      PuplFunc^[q*i+j]:=(n*ot^[q*i+j]+k*PuplFunc^[q*i+j])/z;
     end
   else
   if (n <> 0) then
// если знаки разные и n не нулевой (4 квадрант):
   for i:=q4 to q4+q2-1 do
    for j:=q4 to q4+q2-1 do
     begin
      PuplFunc^[q*i+j]:=-(n*ot^[q*i+j]+k*PuplFunc^[q*i+j])/z;
     end;
end;

var
 i,j,i0: integer;
 c,d0,r: float;
begin
// считываем заданные для данного теста уставки:
 with frmMain do
 begin
  i0:=paracorrInt('sPolSta',sPolSta_Ed.Text); // число звёзд на Lw
// смещение между звёздами в дифр. тестах:
  starShift:=paraCorrEx('Сдвиг звезд',starShift_Ed.Text)*usec/2;
// расстояние между звёздами на изображении поля:
  d0:=paracorrEx('dPolSta',dPolSta_Ed.Text)*StarShift*1e3/Lambda/2;
 end;
// шаг между звёздами на поле, в мм
 c:=Lw/i0/sqrt2;
 DimVector(ot,q*q);
 for i:=-i0 to i0 do
  for j:=-i0 to i0 do
  begin
```

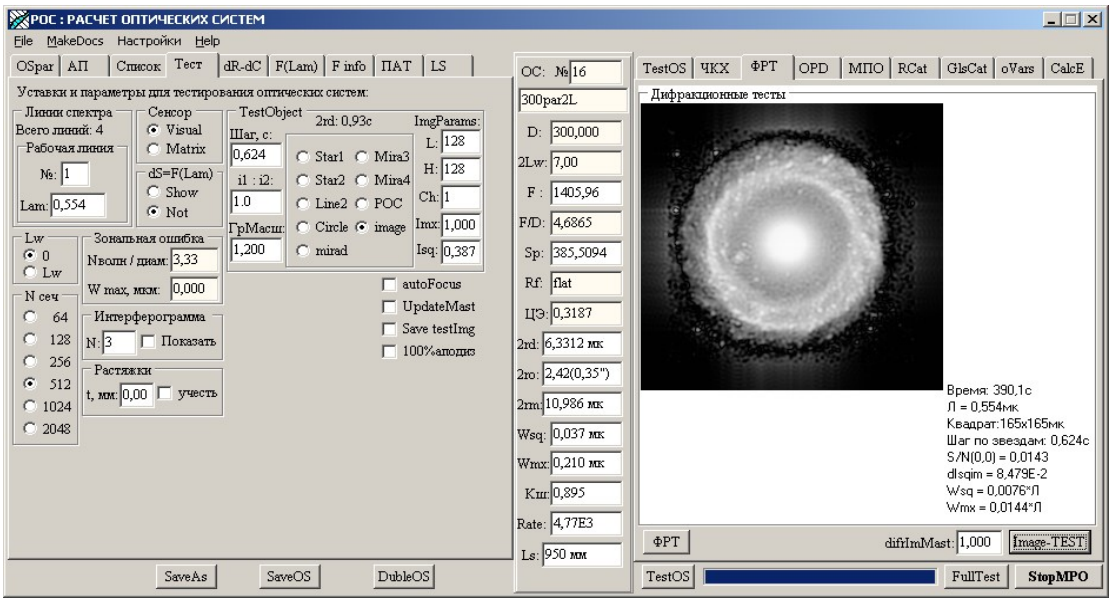
```
   if CalibrImg then
// для эталона - нулевая волновая ошибка:
    ZeroDim(PuplFunc,q)
   else
    begin
// меняя знак у r, разворачиваем зрачковую функцию на 180 градусов:
     r:=-sqrt(sqr(c*i)+sqr(c*j))*sign(i);
     if (i = 0) then
      if (j < 0) then
       r:=-r;
// вычисляем зрачковую функцию для данной точки поля:
     CalcPFv(PuplFunc,r,Lambda,q);
    end;
// преобразуем зрачковую функцию для 1й звезды
// -  в зависимости от её положения на поле:
   BuildPFforPolSta(i,j);
// формируем ФЗ для всего звёздного поля:
   makeArr(1,d0*i,d0*j,0,0);
  end;
  DelVector(ot,q*q);
end;

 // текст РОС формируется из 33 точек
procedure ros;
var b: float;
begin
 b:=dc*sqrt2;
 // буква О
 makeArr(5,1*b,-3*b,-b,b);
 makeArr(5,3*b,-1*b,-b,b);
 makeArr(1,-2*b,2*b,0,0);
 makeArr(1,2*b,-2*b,0,0);
 // буква Р
 makeArr(5,-8*b,-4*b,b,-b);
 makeArr(3,-4*b,-4*b,-b,b);
 makeArr(1,-7*b,-3*b,0,0);
 makeArr(1,-5*b,-5*b,0,0);
 // буква С
 makeArr(5,6*b,2*b,-b,b);
 makeArr(2,5*b,7*b,-b,b);
 makeArr(2,8*b,4*b,-b,b);
 makeArr(1,3*b,7*b,0,0);
 makeArr(1,7*b,3*b,0,0);
end;

// формирование 4х-штриховой миры
procedure mi4;
var
 i:integer;
 b0: float;
begin
 b0:=0.6667*dc;
 i:=18;
 makeArr(i,13*dc,5*dc,0,b0);
```

```
 makeArr(i,9*dc,5*dc,0,b0);
 makeArr(i,5*dc,5*dc,0,b0);
 makeArr(i,-6*dc,-6*dc,-b0,0);
 makeArr(i,-6*dc,-10*dc,-b0,0);
 makeArr(i,-6*dc,-14*dc,-b0,0);
 b0:=b0*0.6667;
 makeArr(i,-10.6*dc,5*dc,-b0,b0);
 makeArr(i,-7.8*dc,7.8*dc,-b0,b0);
 makeArr(i,-5*dc,10.6*dc,-b0,b0);
 makeArr(i,3.8*dc,-11.5*dc,b0,b0);
 makeArr(i,6.6*dc,-14.3*dc,b0,b0);
 makeArr(i,9.4*dc,-17.1*dc,b0,b0);
end;

// формирование 3х-штриховой миры по 5 линий
procedure mi3;
var
 i: integer;
 b0,e0: float;
begin
 b0:=0.43*dc;
 e0:=1.57*b0;
 i:=25;
 makeArr(i,26.5*dc,-9.5*dc,0,2*b0);
 makeArr(i,22.5*dc,-9.5*dc,0,2*b0);
 makeArr(i,18.5*dc,-9.5*dc,0,2*b0);
 makeArr(i,14.5*dc,-9.5*dc,0,2*b0);
 makeArr(i,10.5*dc,-9.5*dc,0,2*b0);
 makeArr(i,-13*dc,4*dc,e0,b0);
 makeArr(i,-15.25*dc,7.25*dc,e0,b0);
 makeArr(i,-17.5*dc,10.5*dc,e0,b0);
 makeArr(i,-19.75*dc,13.75*dc,e0,b0);
 makeArr(i,-22*dc,17*dc,e0,b0);
 makeArr(i,-13*dc,-4*dc,e0,-b0);
 makeArr(i,-15.25*dc,-7.25*dc,e0,-b0);
 makeArr(i,-17.5*dc,-10.5*dc,e0,-b0);
 makeArr(i,-19.75*dc,-13.75*dc,e0,-b0);
 makeArr(i,-22*dc,-17*dc,e0,-b0);
end;

// две линии
procedure Li2;
var a,b: float;
begin
 a:=dc*2; b:=dc/2;
 makeArr(30,a,-15*b,0,b);
 makeArr(30,-a,-15*b,0,b);
end;

 // 4 или 14 точек
procedure star4(i0:integer);
var
 i: integer;
 a,c,d,f,r,s8,c8: float;
```

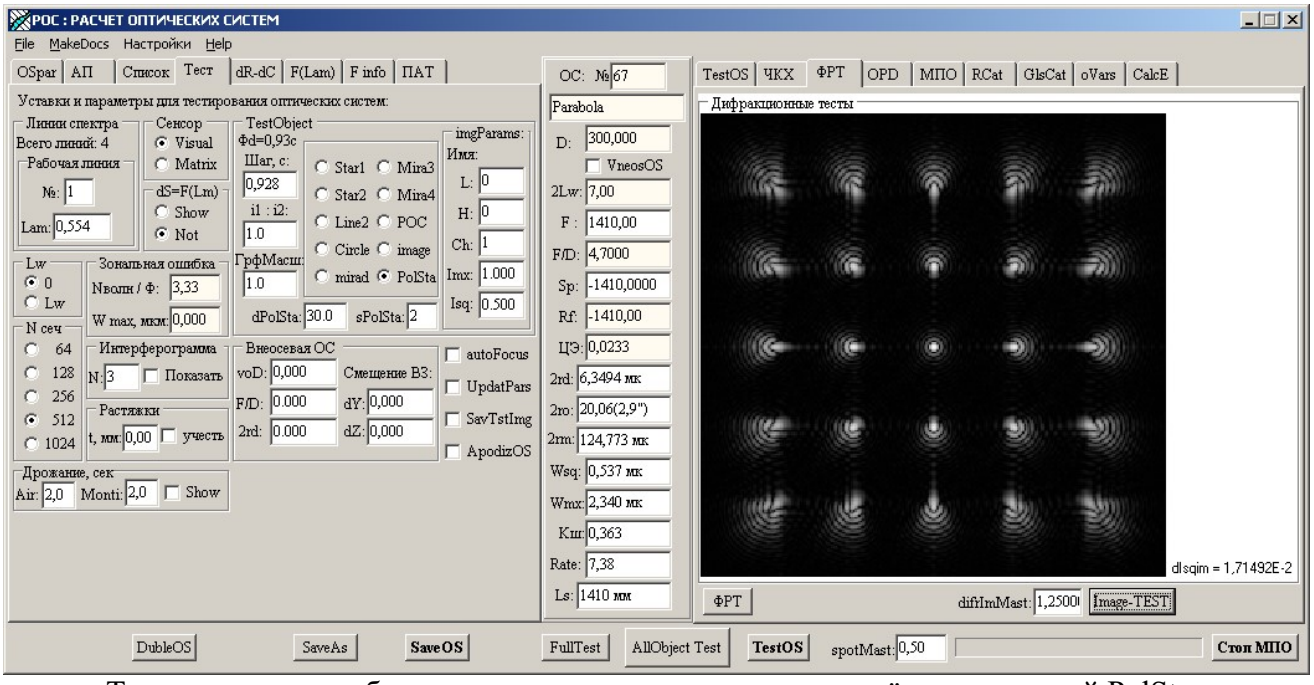
```
begin
 f:=pi/8;
 c8:=cos(f);
 s8:=sin(f);
 for i:=0 to i0 do
 begin
  r:=(4+i)*dc;
  a:=r/sqrt2;
  c:=r*c8;
  d:=r*s8;
  makeArr(1,a,a,0,0);
  makeArr(1,-a,a,0,0);
  makeArr(1,-a,-a,0,0);
  makeArr(1,a,-a,0,0);
  makeArr(1,r,0,0,0);
  makeArr(1,0,r,0,0);
  makeArr(1,-r,0,0,0);
  makeArr(1,0,-r,0,0);
  makeArr(1,c,d,0,0);
  makeArr(1,c,-d,0,0);
  makeArr(1,d,c,0,0);
  makeArr(1,d,-c,0,0);
  makeArr(1,-c,d,0,0);
  makeArr(1,-c,-d,0,0);
  makeArr(1,-d,c,0,0);
  makeArr(1,-d,-c,0,0);
 end;
end;

// 2 звезды
procedure sta2;
var
 ii: integer;
 a: float;
begin
 ii:=round(ParaCorrEx('i1/i2 для двойных
звёзд',frmMain.Ii2zv_Ed.Text));
 a:=2*dc;
 makeArr(1,a,0,0,0);
 makeArr(ii,-a,0,0,0);
end;

// одна звезда
procedure sta1;
begin
 makeArr(1,0,0,0,0);
end;

procedure intrfrnce;
// формирование интерференционной полосы
var
 i,j,n: integer;
 a,y2,z2: float;
begin
```

```
// основноая ФЗ
 makeArr(1,0,0,0,0);
// ФЗ, наклонённая на угол dc*2
 makeArr(1,2*dc,0,0,0);
 for i:=q4 to q4+q2-1 do
 begin
  if i<q2 then
   y2:=sqr(da*(q2-i-0.5))
  else
   y2:=sqr(da*(q2-i+0.5));
  for j:=q4 to q4+q2-1 do
  begin
   if j<q2 then
    z2:=sqr(da*(q2-j-0.5))
   else
    z2:=sqr(da*(q2-j+0.5));
   if ((y2+z2)<=H2) then
// для рабочей зоны ВЗ формируем sin-cos-массивы
   begin
    n:=j+i*q;
    a:=BuildW^[n]*pi2;
    ww1^[n]:=ww1^[n]+cos(a);
    ww2^[n]:=ww2^[n]+sin(a);
   end;
  end;
 end;
end;

begin
 q:=nSec;
 m0:=q*q;
 q2:=q div 2;
 q4:=q div 4;
 da:=Hk*difrImMast/q4;
 if rastajki then
 begin
  qrL:=q2-round(trast/2/da);
  qrR:=q2+round(trast/2/da);
 end
 else
 begin
  qrL:=q2;
  qrR:=q2;
 end;
 // шаг=4*dc=2*Lam*Alf
 dc:=1e3*starShift/Lambda/2;
 if band then
  intrfrnce
 else
  case m of
   0: sta1;
   1: sta2;
   2: Li2;
   3: star4(0);
```

```
    4: star4(16);
    5: mi3;
    6: mi4;
    7: ros;
    9: PolSta;
    else
      Img;
  end;
 end;
end;
```



Пример тестирования системы по снимку кольцевой туманности процедурой Img.



Тестирование параболоида по полю из искусственных звёзд процедурой PolSta.

8. ОПТИМИЗАЦИЯ ПАРАМЕТРОВ

Исходный вариант ОС крайне редко может устроить заказчика, поэтому приходится производить автоматическую коррекцию параметров системы, пытаясь улучшить её характеристики. Для этих целей в разных программах используются различные алгоритмы оптимизации параметров систем, причём авторы расчётных программ обычно ограничиваются каким-то одним, обычно методом наименьших квадратов (МНК), например, Земакс К. Мура, Демос В. Устинова. Д. Дилворс, автор Synopsys, сосредоточился на алгоритме pseudo-second-derivative (PSD), дополнив его «глобальным» методом «отжига» simulation annealing (SimAnn).

В программе РОС используются восемь алгоритмов:
- Lmgold – метод линейного спуска, основанный на «золотом сечении»,
- PostSecDer – использование результатов предыдущих шагов спуска,
- Marquardt – с применением вторых производных оценочной функции,
- SimplexNM – безградиентный алгоритм Нелдера и Мида,
- BFGS – градиентный метод Бройдена-Флетчера-Голдфарба-Шенно,
- GlobalSearch – глобальный метод систематического поиска,
- GeneticAlgo – глобальный «генетический» алгоритм поиска,
- SimulAnneal – глобальный метод «отжига».

По своему определению оптимизация всегда локальна, поскольку призвана выбрать лучшее решение из имеющихся. Кроме того, для оптической системы, имеющей определенный состав (структуру), в узких рамках технического задания в принципе может существовать только локальный минимум. Тем не менее практика доказала целесообразность наличия богатого набора «локальных» алгоритмов оптимизации оптических систем.

Так, локальный метод оптимизации Marquardt, в котором используются, помимо производных первого порядка (градиента), еще и производные второго порядка, эффективен только на начальном этапе оптимизации систем, далёких от совершенства. Буквально за два-три цикла ему удаётся улучшить систему до удовлетворительного уровня. Но дальше он начинает буксовать, так как на равнине вторые производные близки к нулю.

Градиентный метод BFGS по времени и по эффективности наиболее приемлем на среднем этапе оптимизации. Этот универсальный метод – комплексный: в нем определяется градиент, как вектор, указующий направление спуска, но основную работу выполняет программный блок линейной оптимизации LMgold. Несмотря на некоторое замедление, расчёт направляющего градиента вносит в поисковую работу осмысленность действий. Не менее эффективным в BFGS является программный блок, который, используя результаты предыдущей итерации, не вычисляя вторых производных, вносит элемент глобальной оптимизации, делая долгосроч-

ный прогноз процесса оптимизации. К алгоритму BFGS на его входе (в подпрограмме NewGrad) была удачно подключена производная 3-го порядка.

Надо заметить, что BFGS работает «по-крупному», игнорируя нюансы. Так, алгоритм LMgold, с помощью которого BFGS, осуществляет линейный спуск одновременно по всем переменным оптимизации – в отличии, например, от покоординатного спуска. Причём шаг вдоль каждой переменной выбирается пропорционально её производной, что оставляет возможность спрятаться настоящему минимуму вблизи этой срединной траектории.

Алгоритм А. Нелдера и Р. Мида, SimplexNM, благодаря своеобразию и «въедливости», оказывается целесообразным на конечных этапах оптимизации и особенно в тупиковых ситуациях на равнине, когда требуется наивысшая чувствительность к малейшим перепадам в топографии оптимизационной местности.

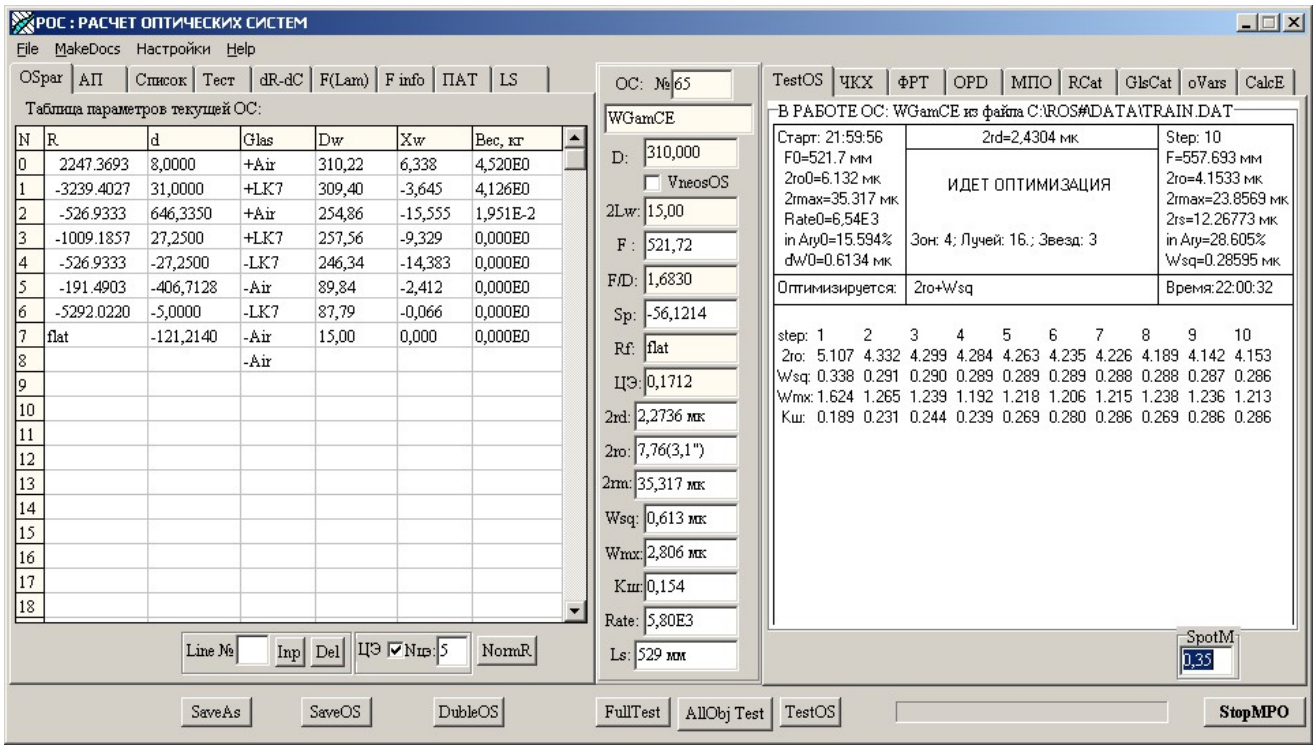
И хотя никакой метод не может выжать из системы более того, на что она способна по своей природе, каждый «локальный» метод позволяет достигать только ему присущий минимум. Причём варьируя параметрами метода, можно всякий раз получать новый минимум, нередко более глубокий, чем предыдущий или достигнутый с помощью других алгоритмов оптимизации. Таким образом, «локальные» методы оптимизации проявляют свои «глобальные» возможности.

Дополнительно к локальным методам в POC можно подключать метод «глобальной» оптимизации, «метод отжига» (SimAnn), обеспечивающий «унифицированное» по всему полю пятно, а также «генетический алгоритм» GeneticAlgo и метод глобального поиска, GlobalSearch.

Формирование скалярного вектора переменных оптимизации

```
procedure insertVars;
var kp: integer;

pu – текущий вектор переменных (толщин, радиусов и т.п.),
np и zp – векторы с номерами переменных в прямом и возвратном ходе,
la – текущий счетчик переменных оптимизации,
kp - глобальный счётчик в списке переменных оптимизации.

procedure insV(var pu:PVector;var np,zp:PbVector;var la:integer);
var i,j,k: integer;
begin
 la:=0;               // начинаем счёт с нуля
 for i:=0 to ns-1 do // сканируем всю ОС, от ВЗ до Фокуса
 begin
  j:=np^[i];          // параметр с номером j отмечен как переменная
  k:=zp^[i];          // номер при возвратном ходе луча
  if ((j=i) and (k=45)) then
// номер параметра I совпал с отмеченным номером j;
// 45 – прочерк, т.е. через параметр нет возвратного хода
  begin
   pX^[kp]:=pu^[i];  // заполняем формальный список переменных
                      // оптимизации значением конкретного параметра
   inc(la);           // увеличиваем текущий счётчик на единицу
   inc(kp);           // увеличиваем на единицу глобальный счётчик
  end;
```

```
 end;
end; {insV}

// формируется массив pX активных параметров:
begin
  kp:=0; // обнуляем глобальный счётчик переменных оптимизации
  insv(rrs,nbb,zbb,abdim);  // добавляем в список радиуса
  insv(ll,nll,zll,aldim);   // добавляем в список толщины
  if asfer then
// если в ОС имеются АП
  begin
   insv(ex,nex,zex,aadim);  // добавляем в список ексцентриситеты
   if asfVP then begin
// если ОС содержит АП-ВП
    insv(a3,nex,zex,aadim); // добавляем в список К3
    insv(a5,nex,zex,aadim); // добавляем в список К5
    insv(a7,nex,zex,aadim); // добавляем в список К7
    insv(a9,nex,zex,aadim); // добавляем в список К9
   end;
  end;
// фиксируем общее число активных переменных оптимизации:
  actiVar:=kp;
end; {insertVars}
```



Программа РОС на 11м шаге оптимизации параметров ОС

sq2ro – это основной программный блок, управлящий ходом оптимизации:
производится изменение значений параметров в списке переменных оптимизации, затем ОС тестируется в новой точке пространства переменных оптимизации.

```
{$F+}
// вычисляет значение целевой функции качества
```

```
// optX – зашифрованный перечень варьируемых переменных
function sq2ro(var optX:pVector): float; {целевая функция качества}
var kp,kpm: integer;
// внесение изменений в параметры ОС по результатам оптимизации:
procedure inpp(var c:char;var np,zp:PbVector;var opu:PVector);
var
 i,j,m,sn: integer;
 b,cr,h: float;
begin
 for i:=0 to ns-1 do
 begin
  j:=np^[i];
  m:=zp^[j];
  if (j = i) then
  begin
   if (m = 45) then
   begin
    if (c = 'r') then
// если оптимизируемый параметр – R или 1/R:
    begin
     cr:=optX^[kp];
     if MPObyC then
// если оптимизация идёт по кривизнам поверхностей:
      if (cr = 0) then
// для плоскости:
       b:=9e9
      else
// для прочих:
       b:=1/cr
     else
// если оптимизация идёт по радиусам поверхностей:
      b:=cr;
// берём значение полудиаметра текущей поверхности:
     h:=abs(hh1^[i])+1;
     if ((abs(b)-h) < 0) then
// если значение предлагаемого радиуса меньше полудиаметра,
// то препятствуем возникновению коллапса:
      b:= h*sign(b);
    end
    else
// иначе – предложение принимается:
     b:=optX^[kp];
    if (c = 'l') then
// если оптимизируемый параметр – толщина:
    begin
     sn:=sign(opu^[i]);
     if (sign(b) <> sn) then
// не позволяем изменять знак у толщин, оставляя его с тем же
// знаком,но почти нулевым, - чтобы была возможность его увеличивать
      b:=eps*q;
    end;
// остальные типы переменных переносятся как есть, без изменений:
    opu^[i]:=b;
    if (kp < kpm) then
```

```
// если не больше числа параметров, то наращиваем глобальный счетчик
     inc(kp);
   end
   else
// если параметр пассивный:
   begin
    b:=opu^[m];
    case c of
    'l': b:=b*sign_d^[i]+ofst_d^[i]; // учитываем знак и смещение
    'r': b:=b*scaleR^[i];            // учитываем масштаб
    'e': b:=b*scaleEx^[i];           // учитываем масштаб
    end;
    pu^[i]:=b; // вставляем изменённое значение параметра
   end;
 end;
end;

// основной блок подпрограммы sq2ro:
var
 ch: char;
 i: integer;
 sum: float;
begin
  kp:=0;       // обнуляем глобальный счётчик
  kpm:=abdim;  // число переменных радиусов
// вносим изменения в разного типа параметры ОС:
  if kpm<>0 then
// если радиуса включены в состав переменных оптимизации
  begin
   ch:='r';    // признак радиуса
   inpp(ch,rrs,nbb,zbb);
  end;
  if aldim<>0 then
// если толщины включены в состав переменных оптимизации
  begin
   ch:='l';    // признак толщины
// наращиваем глобальный счётчик на число переменных толщин
   inc(kpm,aldim);
   inpp(ch,ll,nll,zll);
  end;
  if (asfer and (aadim<>0)) then
  begin
   ch:='e';    // признак АП
   inc(kpm,aadim);
   inpp(ch,ex,nex,zex);
   if KPSh then
// если в системе имеются коррекционные пластины Шмидта:
   begin
    calcH;
    for i:=0 to ns-1 do
     if (i=nex^[i]) then
     begin
      if ((abs(ex^[i])>mizOpt) and (abs(rrs^[i])>=flat)) then
       updateOPK(i);  // обновляем планоидную поверхность
```

```
     end;
   end;
   if asfVP then
// если АП-ВП:
   begin
    inc(kpm,aadim);
    inpp(ch,a3,nex,zex);
    inc(kpm,aadim);
    inpp(ch,a5,nex,zex);
    inc(kpm,aadim);
    inpp(ch,a7,nex,zex);
    inc(kpm,aadim);
    inpp(ch,a9,nex,zex);
   end;
  end;
{вычисление функции качества при новых параметрах ОС; minro=1e-5}
   CRate;     // вычислили характеристики качества обновлённой ОС
   sum:=0;    // обнуляем значение функции качества
// и вычисляем её значение при новых параметрах ОС:
   if b0 then // если установлена опция «стабилизировать фокус»
    sum:=sum+sqr((Fp-Fl)*wesF);
   if b1 then // если нужно стабилизировать последний отрезок
    sum:=sum+sqr((abs(ll^[ns-1])-LastD)*wesSp);
   if b2 then // если нужно уменьшить дрожание до кружка Эйри
    sum:=sum+sqr((Fl*Droj-rdfr)*wesDrj);
   if b3 then // если нужно уменьшить пятно до кружка Эйри
    sum:=sum+sqr((srod-minro)*wes2ro);
   if b4 then // если нужно уменьшить 2romax до кружка Эйри
    sum:=sum+sqr((maxRo-rdfr)*wes2rmx);
   if b5 then // если нужно уменьшить dWsq до minro
    sum:=sum+sqr((swar-minro)*wesWsqv);
   if b6 then // если нужно уменьшить dWmax до minro
    sum:=sum+sqr((swam-minro)*wesWmx);
   if b7 then // если нужно увеличить рейтинг до Ratemax
    sum:=sum+sqr((maxRate-Rate)*wesRate);
   if b8 then // если нужно увеличить псевдо-Кш до 1
    sum:=sum+sqr((1-sumAry)*wesKsh);
   inc(Nsq2ro); // число вычислений sq2ro
   result:=sqrt(sum); // значение целевой функции качества
end;
{$F-}

{=============================================================}
```

Далее слеует программный код (процедур F_r, LMgold, CalcGrad, BFGS и др.), заимствованный из пакета программных модулей на Паскале, выложенного Jean Debord в сети Internet для свободного пользования, и адаптированный к особенностям программы РОС.

```
{ проба целевой функции в точке с приращением формальных параметров на r*DeX,
 где DeX – значение градиента (первой производной от целевой функции по параметру pX)}

  function F_r(r:float):float;
  var i: integer;
```

```
  begin
   for i:=0 to nvar do
    Xt^[i]:=pX^[i]+r*DeX^[i];
   F_r:=sq2ro(Xt);
  end;
```

{=== алгоритм линейного спуска с применением «золотого сечения»:
модифицирует переменную оптимизации до 1000 * Gradient
GOLD = 1.61803398874989484821, CGOLD = 1 – GOLD ==========}

```
procedure LMgold;
var
 A,B,C,Fa,Fb,Fc,F1,F2,X0,X1,X2,X3: Float;
 k: integer;
{_______ проба целевой функции в точке «золотого сечения» ______}
 procedure MB2;
 begin
  C:=B+GOLD*(B-A);
  Fc:=F_r(C);
 end;
{_____________________________________________________________________}
begin
 A:=0;
 B:=1;
 Fa:=fu;
 Fb:=F_r(B);
 if Fb>Fa then
 begin
  FSwap(A,B);
  FSwap(Fa,Fb);
 end;
 MB2;
 while Fc<Fb do
 begin
  A:=B;
  B:=C;
  Fa:=Fb;
  Fb:=Fc;
  MB2;
 end;
 if A>C then
 begin
  FSwap(A,C);
  FSwap(Fa,Fc);
 end;
 X0:=A;
 X3:=C;
 if (C-B)>(B-A) then
 begin
  X1:=B;
  F1:=Fb;
  X2:=B+CGOLD*(C-B);
  F2:=F_r(X2);
 end
```

```
 else
 begin
  X2:=B;
  F2:=Fb;
  X1:=B-CGOLD*(B-A);
  F1:=F_r(X1);
 end;
 k:=1;
// GoldIter = 100, precesion = eps = 1.65e-10.
 while ((k<GoldIter) and (abs(X3-X0)>precesion*(abs(X1)+abs(X2))))
do
  if F2<F1 then
  begin
   X0:=X1;
   X1:=X2;
   F1:=F2;
   X2:=X1+CGOLD*(X3-X1);
   F2:=F_r(X2);
   inc(k);
  end
  else
  begin
   X3:=X2;
   X2:=X1;
   F2:=F1;
   X1:=X2-CGOLD*(X2-X0);
   F1:=F_r(X1);
   inc(k);
  end;
  if F1<F2 then
   fu:=F_r(X1)
  else
   fu:=F_r(X2);
// переносим полученные значения переменных из Xt в pX:
  copyVector(pX,Xt,0,nvar);
end;

{=============== вычисление градиента ================}

procedure CalcGrad;
var
 t,dlx,Fpls,Fmns: Float;
 i: Integer;
begin
 for i:=0 to nvar do
 begin
  t:=pX^[i];
  if t<>0 then // в работе участвуют только ненулевые переменные
  begin
   dlx:=delPar*t;   // образуем приращение переменной
   pX^[i]:=t-dlx;   // уменьшаем переменную
   Fmns:=sq2ro(pX); // вычисляем целевую функцию
   pX^[i]:=t+dlx;   // увеличиваем переменную
   Fpls:=sq2ro(pX); // вычисляем целевую функцию
```

```
// Получаем значение градиента с удвоенной точностью:
   pG^[i]:=(Fpls-Fmns)/(dlx*2);
   pX^[i]:=t;        // возвращаем переменной исходное значение
  end
  else
   pG^[i]:=0;        // если переменная нулевая
 end;
end;

{=====================================================================}
// В отличие от CalcGrad здесь используется производная 3го порядка:
{=====================================================================}
procedure NewGrad;
var
 a,d,pr1,pr2,pr3,t,dlx,Fpls,Fmns: Float;
 i: Integer;
begin
 for i:=0 to nvar do
 begin
  t:=pX^[i];
  if t<>0 then
  begin
   dlx:=delPar*t;
   pX^[i]:=t-dlx;
   Fmns:=sq2ro(pX);
   pX^[i]:=t+dlx;
   Fpls:=sq2ro(pX);
   pr1:=(Fpls-Fmns)/(dlx*2);
   pr2:=(Fpls-fu*2+Fmns)/sqr(dlx);
   pG^[i]:=pr1;
   if abs(pr2)>eps then
   begin
    a:=-pr1/pr2;
    if fu>Rdfr then
     d:=sqr(a)-Rdfr*2/pr2
    else
     d:=sqr(a)-fu*2/pr2;
    if d>0 then
     DeX^[i]:=a+sqrt(d)
    else
     DeX^[i]:=a;
    if pr1<usec then
    begin
     pX^[i]:=t+dlx*2;
// вычисление производной 3го порядка:
     pr3:=(sq2ro(pX)-Fpls*3+Fmns*3-fu)/dlx/sqr(dlx);
     if pr3<>0 then
      DeX^[i]:=-pr2*2/pr3
     else
      DeX^[i]:=pr1;
    end;
   end
   else
    DeX^[i]:=-pr1;
```

```
   pX^[i]:=t;
  end
  else
   DeX^[i]:=0;
 end;
end;

{====== алгоритм Бройдена-Флетчера-Голдфарба-Шенно ======}

procedure BFGS;
var
 i,j,itr,n: integer;
 sum1,sum2,r1,r2: float;
begin
 n:=nvar;
 NewGrad; //
 if AbsMax(pG,n)<ibm0 then
// если максимальный градиент ничтожно мал, выходим из процедуры:
 begin
  ConOpt:=true;
  exit;
 end;
// формируем начальную матрицу-псевдогессиан:
 for i:=0 to n do
  for j:=0 to n do
   if i=j then
    pH^[i]^[j]:=1
   else
    pH^[i]^[j]:=0;
 itr:=1;
 repeat
  if qut then
// если пользователь прервал процесс, выходим из процедуры:
  begin
   ConOpt:=true;
   exit;
  end;
  mxm:=AbsMax(DeX,n);
  if (mxm>2) then
// если градиенты велики, нормируем их по максимуму:
   for i:=0 to n do
    DeX^[i]:=DeX^[i]/mxm;
// сохраняем предыдущие значения пременных и градиентов:
  CopyVector(OldX,pX,0,n);
  CopyVector(OldG,pG,0,n);

// делаем линейный спуск:
  LMSearch;
// вычисляем градиенты при новых значениях переменных:
  CalcGrad;
// вычисляем приращения градиентов и переменных оптимизации:
  for i:=0 to n do
    begin
     dG^[i]:=pG^[i]-OldG^[i];
```

```
     dX^[i]:=pX^[i]-OldX^[i];
    end;
// вычисляем вспомогательные величины:
  for i:=0 to n do
  begin
    sum1:=0;
    for J:=0 to n do
     sum1:=sum1+pH^[i]^[J]*dG^[J];
    HdG^[i]:=sum1;
  end;
  sum1:=0;
  sum2:=0;
  for i:=0 to n do
  begin
   r1:=dG^[i];
   sum1:=sum1+r1*dX^[i];
   sum2:=sum2+r1*HdG^[i];
  end;
  if ((sum1<>0) and (sum2<>0)) then
  begin
    for i:=0 to n do
    begin
      R1dX^[i]:=dX^[i]/sum1;
      R2HdG^[i]:=HdG^[i]/sum2;
      pU^[i]:=R1dX^[i]-R2HdG^[i];
      P2U^[i]:=sum2*pU^[i];
    end;
    for i:=0 to n do
    begin
      r1:=r1dx^[i];
      r2:=r2hdg^[i];
      sum2:=p2u^[i];
// модифицируем псевдо-гессиан:
      for j:=0 to n do
       pH^[i]^[j]:=pH^[i]^[j]+r1*dX^[j]-r2*HdG^[j]+sum2*pU^[j];
    end;
// вычисляем аналитические значения градиентов:
    for i:=0 to n do
    begin
      sum1:=0;
      for j:=0 to n do
       sum1:=sum1-pH^[i]^[j]*pG^[j];
      DeX^[i]:=sum1;
    end;
  end;
  Inc(itr);
  Application.ProcessMessages;
 until ((itr>=itMaxBfgs) or conOpt);
end;


{========== алгоритм PSD ==========}

procedure PSD;
```

```
// post-second derivative algorithm:
var
 i,j,itr,n: integer;
 deg: float;
begin
 n:=nvar;
// входим в процедуру с новыми значениями градиента:
 NewGrad;
 itr:=1;
 repeat
  if AbsMax(pG,n)<ibm0 then
   qut:=true;
  if qut then
  begin
   ConOpt:=true;
   exit;
  end;
// Сохраняем старые значения переменных и градиентов:
  CopyVector(OldX,pX,0,n);
  CopyVector(OldG,pG,0,n);
// если градиент велик, нормируем по максимуму:
  mxm:=AbsMax(DeX,n);
  if (mxm>1) then
   for i:=0 to n do
    DeX^[i]:=DeX^[i]/mxm;
// делаем линейный спуск по целевой функции:
  LMSearch;
// вычисляем новый градиент:
  CalcGrad;
// прогнозируем градиент, учитывая результаты оптимизации:
  for i:=0 to n do
  begin
   deg:=pG^[i]-OldG^[i];
   if deg=0 then
    DeX^[i]:=0
   else
     DeX^[i]:=-pG^[i]*abs(pX^[i]-OldX^[i])/deg;
  end;
  Inc(itr);
// посыл на обновление ситуации в операционной системе:
  Application.ProcessMessages;
 until ((itr >= itMaxBFGS) or conOpt);
end;
```

{========== симплекс-алгоритм А. Нелдера и Р. Мида ==========}

```
  procedure Simplex;
  var
   iH,iL: integer;

// обновление симплекс-матрицы:
   procedure UpdateSimplex(Y:Float;pQ:PVector);
    begin
      pF^[iH]:=Y;
```

```
      CopyVector(pp^[iH],pQ,0,nVar);
    end;

  var
   Flag: Boolean;
   corr,maxCorr,f0,sum,Ystar,Y2star: Float;
   i,j,iter,m: Integer;
  begin
    m:=nVar+1;
    Iter:=1;
    F0:=9e9;
// сохраняем старые значения переменных:
    for i:=0 to m do
      CopyVector(pp^[i],pX,0,nVar);
{ формируем симлекс-матрицу с новыми значениями переменных. Коэффи-
циент simplexStep определяет диапазон варьирования переменных:}
    for I :=0 to nVar do
      pp^[i]^[i]:=pp^[i]^[i]*simplexStep;
// вычисляем целевые функции при новых переменных:
    for i:=0 to m do
     pF^[i]:=sq2ro(pp^[i]);
    repeat
      iL:=0;
      iH:=0;
// находим номера с мин. и макс. значениями целевой функции:
      for i:=1 to m do
        if (pF^[I] < pF^[iL]) then
          iL:=I
        else
         if (pF^[I] > pF^[iH]) then
          iH:=I;
      if (pF^[iL] < F0) then
        F0:=pF^[iL];
// вычисляем вспомогательные величины и обновляем симплекс-матрицу:
      for J:=0 to nVar do
        begin
          Sum:=0;
          for I:=0 to m do
            if (i <> iH) then
             Sum:=Sum+pp^[I]^[J];
          Pbar^[j]:=Sum/m;
        end;
      for j:=0 to nVar do
        Pstar^[J]:=2*Pbar^[J]-pp^[iH]^[J];
      Ystar:=sq2ro(Pstar);
      if (Ystar < pF^[iL]) then
        begin
          for J:=0 to nVar do
            P2star^[J]:=Pstar^[J]*3-Pbar^[J]*2;
          Y2star:=sq2ro(P2star);
          if (Y2star < pF^[iL]) then
            UpdateSimplex(Y2star,P2star)
          else
            UpdateSimplex(Ystar,Pstar);
```

```
        end
      else
        begin
          I:=0;
          Flag:=False;
          repeat
            if ((I <> iH) and (pF^[I] > Ystar)) then
             Flag:=True;
            Inc(I);
          until (Flag or (I > m));
          if Flag then
            UpdateSimplex(Ystar,Pstar)
          else
            begin
              if (Ystar <= pF^[iH]) then
                UpdateSimplex(Ystar,Pstar);
              for J:=0 to nVar do
                P2star^[J]:=(pp^[iH]^[J]+Pbar^[J])/2;
              Y2star:=sq2ro(P2star);
              if (Y2star <= pF^[iH]) then
                UpdateSimplex(Y2star,P2star)
              else
                for i:=0 to m do
                  for J:=0 to nVar do
                    pp^[i]^[J]:=(pp^[i]^[J]+pp^[iL]^[J])/2;
            end;
        end;
      MaxCorr:=0;
      for J:=0 to nVar do
        begin
          Corr:=Abs(pp^[iH]^[J]-pp^[iL]^[J]);
          if (Corr > MaxCorr) then
           MaxCorr:=Corr;
        end;
      Inc(Iter);
    until ((MaxCorr < eps) or (Iter > itMaxSimp));
// переносим из симплекс-матрицы в pX значения новых переменных:
    CopyVector(pX,pp^[iL],0,nVar);
// вычисляем новое значение целевой функции:
    fu:=pF^[iL];
  end;
```

Процедура вычисления гессиана, содержащего градиенты и 2е производные целевой функции по параметрам оптимизации.

```
  {$F+}
  procedure NumHessGrad;
  var
    Delta,Xminus,Xplus,Fminus,Fplus: PVector;
    Temp1,Temp2,F,F2plus: Float;
    I,J: Integer;
  Begin
// формируем рабочие динамические массивы:
    DimVector(Delta,nVar);
```

```
    DimVector(Xminus,nVar);
    DimVector(Xplus,nVar);
    DimVector(Fminus,nVar);
    DimVector(Fplus,nVar);
// вычисляем исходное значение целевой функции:
    F:=sq2ro(pX);
// для каждой переменной вычисляем приращение и её значение:
    for I:=0 to nVar do
    begin
     if (pX^[I] <> 0) then
      Delta^[I]:=Abs(pX^[I])*eps
     else
      Delta^[I]:=eps;
     Xplus^[I]:=pX^[I]+Delta^[I];
     Xminus^[I]:=pX^[I]-Delta^[I];
    end;
// вычисляем значения целевой функции для +/- значений переменных:
    for I:=0 to nVar do
    begin
     Temp1:=pX^[I];
     pX^[I]:=Xminus^[I];
     Fminus^[I]:=sq2ro(pX);
     pX^[I]:=Xplus^[I];
     Fplus^[I]:=sq2ro(pX);
     pX^[I]:=Temp1;
    end;
// вычисляем 1е и 2е производные от целевой функции по переменным:
    for I:=0 to nVar do
      begin
        pG^[I]:=(Fplus^[I]-Fminus^[I])/2/Delta^[I];
        pH^[I]^[I]:=(Fplus^[I]+Fminus^[I]-F*2)/Sqr(Delta^[I]);
      end;
// вычисляем частные производные 2го порядка и строим гессиан:
    for I:=0 to nVar-1 do
    begin
     Temp1:=pX^[I];
     pX^[I]:=Xplus^[I];
     for J:=Succ(I) to nVar do
     begin
       Temp2:=pX^[J];
       pX^[J]:=Xplus^[J];
       F2plus:=sq2ro(pX);
     pH^[I]^[J]:=(F2plus-Fplus^[I]+Fplus^[J]+F)/Delta^[I]/Delta^[J];
       pH^[J]^[I]:=pH^[I]^[J];
       pX^[J]:=Temp2;
     end;
     pX^[I]:=Temp1;
    end;
// освобождаем память от массивов:
    DelVector(Delta,nVar);
    DelVector(Xminus,nVar);
    DelVector(Xplus,nVar);
    DelVector(Fminus,nVar);
    DelVector(Fplus,nVar);
```

```
  end;
  {$F-}
```

Процедура ParamConv проверяет завершение процесса оптимизации, когда приращения переменных становятся ничтожно малыми.

```
  function ParamConv: Boolean;
  var
    i: Integer;
    Conv: Boolean;
    a: float;
  begin
    i:=0;
    Conv:=True;
    repeat
     a:=OldX^[i];
// проверка приращения переменных оптимизации на «мизерность»:
     Conv:=(Conv and (Abs(pX^[i]-a) < FMax(eps,eps*Abs(a))));
     Inc(i);
    until ((Conv=False) or (i > nVar));
    ParamConv:=Conv;
  end;
```

// ========== алгоритм Марквардта ==========

```
  procedure Marquardt(canvas:Tcanvas;OSparGr,ASurfGrd:TStringGrid);
  var
    Lam,Lam1   : Float;    { Marquardt's lambda }
    i,j        : Integer;  { Loop variable }
    Det,F_min  : Float;    { Determinant of H }
    F1         : Float;    { New minimum }
    Lambda_Ok  : Boolean;  { Successful Lambda decrease }
    Conv       : Boolean;  { Convergence reached }
    Done       : Boolean;  { Iterations done }
    ErrCode    : Integer;  { Error code }
    Lbound, Ubound: integer;
  begin
    Lam:=marqLamb0;   // даём множителю начальное значени
    F_min:=sq2ro(pX); // начальное значение целевой функции
    LBound:=0;
    Ubound:=nVar;
    j:=0;             // обнуляем счетчик для вывода информации
    Conv:=False;
    Done:=False;
    Repeat
//сохраняем старое значение переменных оптимизации:
      CopyVector(OldX,pX,Lbound,Ubound);
// вычисляем гессиан:
      NumHessGrad;
// изменяем знаки градиентов на противоположные:
      for i:=Lbound to Ubound do
        pG^[i]:=-pG^[i];
      if Conv then  { Newton-Raphson iteration }
        begin
```

```
// если переменные уже не изменяются, сохраняем гессиан и градиент:
        CopyMatrix(pp,pH,Lbound,Lbound,Ubound,Ubound);
        CopyColFromVector(pp,pG,Lbound,Ubound,Succ(Ubound));
// вычисляем определитель гессиана:
        ErrCode:=GaussJordan(pp,Lbound,Ubound,Succ(Ubound),Det);
        if (ErrCode = MAT_OK) then
          begin
// если ОК, сохраняем градиент и вычисляем новые переменные:
            CopyVectorFromCol(DeX,pp,Lbound,Ubound,Succ(Ubound));
            for i:=Lbound to Ubound do
              pX^[i]:=OldX^[i]+DeX^[i];
          end;
        Done:=True;
      end
    else          { Marquardt iteration }
      begin
        repeat
// сохраняем гессиан и градиент:
          CopyMatrix(pp,pH,Lbound,Lbound,Ubound,Ubound);
          CopyColFromVector(pp,pG,Lbound,Ubound,Succ(Ubound));
// увеличиваем множитель на единицу:
          Lam1:=1+Lam;
// увеличиваем диагональ гессиана:
          for i:=Lbound to Ubound do
            pp^[i]^[i]:=Lam1*pH^[i]^[i];
          Lambda_OK:=False;
// вычисляем определитель гессиана:
          ErrCode:=GaussJordan(pp,Lbound,Ubound,Succ(Ubound),Det);
          if (ErrCode = MAT_OK) then
            begin
// если ОК, сохраняем новые значения переменных и градиента:
             CopyVector(pX,OldX,Lbound,Ubound);
             CopyVectorFromCol(DeX,pp,Lbound,Ubound,Succ(Ubound));
// делаем линейный спуск по целевой функции:
              LMsearch;
              F1:=fu;
// проверяем, насколько велик спуск по целевой функции:
              Lambda_Ok:=((F1-F_min) < F_min*eps);
              if (not Lambda_Ok) then
// если спуск мал, увеличиваем множитель:
               Lam:=Lam*marqKincdwn;
              if (Lam > marqLambMax) then
// если множитель превысил максимальное значение, сообщаем об этом:
               ErrCode:=OPT_BIG_LAMBDA;
            end;
        until (Lambda_Ok or (ErrCode <> MAT_OK));
// проверяем переменные на изменяемость:
        Conv:=ParamConv;
// уменьшаем множитель:
        Lam:=Lam/marqKincdwn;
        F_min:=F1;
        inc(j);
        if (j=50) then
        begin
```

```
// если пора, выводим информацию о ходе оптимизации на экран:
          Loop(canvas,OSparGr,ASurfGrd);
          j:=0;
         end;
      end;
    until (qut or Done or (ErrCode <> OPT_OK));
    fu:=F_min;
  end;
```

// ==========Метод имитации отжига SimAnn ==========

Несколько вспомогательных процедур для вычисления случайных чисел:

```
  procedure RMarIn(Seed1, Seed2 : Integer);
  begin
    XR1:=Seed1;
    XR2:=Seed2;
    C1:=0;
    C2:=0;
  end;

  function IRanMar: LongInt;
  var
    Y1,Y2: LongInt;
  begin
    Y1:=18000*XR1+C1;
    XR1:=Y1 and 65535;
    C1:=Y1 shr 16;
    Y2:=30903*XR2 + C2;
    XR2:=Y2 and 65535;
    C2:=Y2 shr 16;
    IRanMar:=(XR1 shl 16)+(XR2 and 65535);
  end;

  function RanMar: Float;
  begin
    RanMar:=(IRanMar+2147483648.0)/4294967296.0;
  end;
```

Процедура быстрой сортировки чисел по убыванию значений:

```
  procedure QSort(X:PVector;Lbound,Ubound:Integer);
    procedure Sort(L,R:Integer);
    var
      I,J: Integer;
      U,V: Float;
    begin
      I:=L;
      J:=R;
      U:=X^[(L+R) div 2];
      repeat
        while (X^[I] < U) do
         inc(i);
```

```
        while (U < X^[J]) do
         dec(j);
        if (I <= J) then
          begin
            V:=X^[I];
            X^[I]:=X^[J];
            X^[J]:=V;
            inc(i);
            dec(j);
          end;
      until (I > J);
      if (L < J) then
       Sort(L,J);
      if (I < R) then
       Sort(I, R);
    end;
  begin
    Sort(Lbound,Ubound);
  end;

// --------- выделение из массива медианного значения ---------

  function Median(X:PVector;Lbound,Ubound:Integer): Float;
  var
    N,N2: Integer;
  begin
    N:=Ubound-Lbound+1;
    N2:=(N div 2)+Lbound-1;
    QSort(X,Lbound,Ubound);
    if Odd(N) then
      Median:=X^[N2+1]
    else
      Median:=(X^[N2]+X^[N2+1])/2;
  end;
```

Процедура InitTempr вычисляет массив из N_Eval случайных значений целевой функции и принимает в нём медианное значение за исходную величину при поиске минимума.

```
  function InitTempr: Float;
  var
    F,F1: Float;
    i,k,N_inc : Integer;
  begin
    N_inc:=0;
    F:=sq2ro(pX);
    K:=0;
    for I:=1 to N_EVAL do
      begin
// вычисляем целевую функцию для случайного значения переменной:
        pX^[K]:=Xmn^[K]+RanMar*Range^[K];
        F1:=sq2ro(pX);
        if (F1 > F) then
```

Если целевая функция возросла, увеличиваем счётчик N_inc и запоминаем приращение целевой функции:

```
          begin
            Inc(N_inc);
            DeltaF^[N_inc]:=F1-F;
          end;
```

Если целевая функция уменьшилась, меняем исходное значение на уменьшенное:

```
        F:=F1;
        Inc(K);
        if (K > nVar) then
         K:=0;
      end;
    if (N_inc > 0) then
```

Если целевая функция возрастала, то из массива выбираем медианное значение:

```
      InitTempr:=Median(DeltaF,1,N_inc)/LN2
    Else
```

Если целевая функция не возрастала:

```
      InitTempr:=1;
  end;
```

Процедура принимает решение, допустимо ли приращение целевой функции:

```
  function Accept(dF,T:Float;var N_inc,N_acc:Integer): Boolean;
  begin
    if (dF < 0) then
      Accept:=True
    else
      begin
        Inc(N_inc);
        if (Exp(-dF/T) > RanMar) then
          begin
            Accept:=True;
            Inc(N_acc);
          end
        else
          Accept:=False;
      end;
  end;

  function SAparConv(X,Stp:PVector):Boolean;
  var
    I: Integer;
    Conv: Boolean;
  begin
    I:=0;
    Conv:=True;
    repeat
```

```
    Conv:=(Conv and (Stp^[I] < FMax(eps,eps*Abs(X^[I]))));
    Inc(I);
  until ((Conv = False) or (I > nVar));
  SAparConv:=Conv;
end;


// программный код «глобального» алгоритма «имитации отжига»
procedure SimAnn(canvas:Tcanvas;OSparGr,ASurfGrd:TStringGrid);
var
  I,J,K,n,N_inc,N_acc,Lbound,Ubound : Integer;
  sum,dv,F,F1,F_min,dF,Ratio,T,oX : Float;
  Stp: PVector;
  Nacc: PIntVector;
begin
  LBound:=0;
  Ubound:=nVar;
  DimVector(Stp,Ubound);
  DimIntVector(Nacc,Ubound);
  for i:=0 to nvar do
  begin
   dv:=abs(dVsimAnn*pX^[i])/2;
   Xmn^[i]:=pX^[i]-dv;
   Xmx^[i]:=pX^[i]+dv;
   Range^[i]:=dv*2;
   Stp^[i]:=dv;
   Xt^[i]:=pX^[i];
  end;
  F:=sq2ro(pX);
  F_min:=F;
  T:=InitTempr;
  n:=0;
  repeat
    N_inc:=0;
    N_acc:=0;
    sum:=0;
    for i:=1 to SA_Nt do
      begin
        for J:=1 to SA_Ns do
          for K:=Lbound to Ubound do
            begin
              oX:=pX^[K];
              pX^[K]:=pX^[K]+(RanMar*2-1)*Stp^[K];
              if ((pX^[K] < Xmn^[K]) or (pX^[K] > Xmx^[K])) then
                pX^[K]:=Xmn^[K]+RanMar*Range^[K];
              F1:=sq2ro(pX);
              dF:=F1-F;
              if Accept(dF,T,N_inc, N_acc) then
                begin
                  Inc(Nacc^[K]);
                  F:=F1;
                end
              else
                pX^[K]:=oX;
```

```
                if (F < F_min) then
                  begin
                    Xt^[K]:=pX^[K];
                    F_min:=F;
                  end;
              end;
            for K:=Lbound to Ubound do
              begin
                Ratio:=Int(Nacc^[K])/Int(SA_Ns);
                sum:=sum+Ratio;
                if (Ratio > 0.6) then
                  begin
                    Stp^[K]:=Stp^[K]*(1+((Ratio-0.6)/0.4)*N_FACT);
                    if (Stp^[K] > Range^[K]) then
                     Stp^[K]:=Range^[K];
                  end
                else
                 if (Ratio < 0.4) then
                  Stp^[K]:=Stp^[K]/(1+((0.4-Ratio)/0.4)*N_FACT);
                Nacc^[K]:=0;
              end;
          end;
        if (sum/SA_Nt/SA_Ns/nVar <=0.9) then
         T:=T*SA_Rt;
        if SAparConv(Xt,Stp) then
         ConOpt:=true;
        inc(n);
        if (n=12) then
        begin
         tSimAnn:=T;
         fu:=F_min;
         Loop(canvas,OSparGr,ASurfGrd);
         n:=0;
        end;
        Application.ProcessMessages;
      until (qut or conOpt);
      for K:=Lbound to Ubound do
        pX^[K]:=Xt^[K];
      fu:=sq2ro(pX);
      DelVector(Stp,Ubound);
      DelIntVector(Nacc,Ubound);
    end;

{====================== Genetic Algo ==========================}

procedure GenAlg(canvas:Tcanvas;Grd0,Grd1:TStringGrid);
var
 pC1,pC2,pD,pM: PMatrix;
 Fn: PVector;

procedure Mutate(i:Integer);
var
 J: Integer;
begin
```

```
 for J:=0 to nVar do
 begin
  pC1^[i]^[J]:=Xmn^[J]+RanMar*Range^[J];
  pC2^[i]^[J]:=Xmn^[J]+RanMar*Range^[J];
  pD^[i]^[J]:=RanMar;
  pM^[i]^[J]:=pD^[i]^[J]*pC1^[i]^[J]+(1-pD^[i]^[J])*pC2^[i]^[J];
 end;
end;

procedure Cross(i1,i2,i:Integer);
var
 J,K: Integer;
begin
 for J:=0 to nVar do
 begin
  if (RanMar < 0.5) then
   K:=i1
  else
   K:=i2;
  pC1^[i]^[J]:=pC1^[K]^[J];
  if (RanMar < 0.5) then
   K:=i1
  else
   K:=i2;
  pC2^[i]^[J]:=pC2^[K]^[J];
  pD^[i]^[J]:=RanMar;
  pM^[i]^[J]:=pD^[i]^[J]*pC1^[i]^[J]+(1-pD^[i]^[J])*pC2^[i]^[J];
 end;
end;

procedure Homozygote(i:Integer);
begin
 CopyVector(pC1^[i],pM^[i],0,nVar);
 CopyVector(pC2^[i],pM^[i],0,nVar);
end;

function GA_Func(i:Integer): float;
begin
 CopyVector(Xt,pM^[i],0,nVar);
 GA_Func:=sq2ro(Xt);
end;

procedure CompFunc;
var
  i,J,K: Integer;
  A: Float;
begin
 if qut then
  exit;
 for i:=0 to GA_NP-1 do
  Fn^[i]:=GA_Func(i);
 for i:=0 to GA_NP-1 do
 begin
  K:=i;
```

```
  A:=Fn^[i];
  for J:=i+1 to GA_NP-1 do
   if (Fn^[J] < A) then
   begin
    K:=J;
    A:=Fn^[J];
   end;
   FSwap(Fn^[i],Fn^[K]);
   for J:=0 to nVar do
   begin
    FSwap(pC1^[i]^[J],pC1^[K]^[J]);
    FSwap(pC2^[i]^[J],pC2^[K]^[J]);
    FSwap(pD^[i]^[J],pD^[K]^[J]);
    FSwap(pM^[i]^[J],pM^[K]^[J]);
   end;
  end;
  if (Fn^[0] < fu) then
  begin
   fu:=Fn^[0];
   CopyVector(pX,pM^[0],0,nVar);
  end;
end;

procedure GenPop(NS:integer);
var
 i,i1,i2: integer;
 F0: Float;
begin
 if qut then
  exit;
 for i:=NS+1 to GA_NP-1 do
 begin
  i1:=Trunc(RanMar*NS)+1;
  repeat
   i2:=Trunc(RanMar*NS)+1
  until (i2 <> i1);
  F0:=FMax(Fn^[i1],Fn^[i2]);
  repeat
   Cross(i1,i2,i);
  until (GA_Func(i) <= F0);
 end;
 for i:=0 to GA_NP-1 do
 begin
  if (RanMar < GA_MR) then
   Mutate(i);
  if (RanMar < GA_HR) then
   Homozygote(i);
 end;
end;

var
  i,NS: Integer;
  f0: float;
begin
```

```
  DimMatrix(pC1,GA_NP,nVar);
  DimMatrix(pC2,GA_NP,nVar);
  DimMatrix(pD,GA_NP,nVar);
  DimMatrix(pM,GA_NP,nVar);
  DimVector(Fn,GA_NP);
  f0:=fu;
  for i:=0 to nVar do
  begin
    Xmn^[i]:=pX^[i]*(1-vRange);
    Xmx^[i]:=pX^[i]*(1+vRange);
    Range^[i]:=Xmx^[i]-Xmn^[i];
  end;
  NS:=Trunc(GA_NP*GA_SR);
  nGraFun:=0;
  for i:=0 to GA_NP-1 do
    Mutate(i);
  CompFunc;
  for i:=0 to GA_NG-1 do
  begin
   if qut then
    break;
   Loop(canvas,Grd0,Grd1);
   GenPop(NS);
   CompFunc;
   fu:=sq2ro(pX);
   if (fu < f0) then
   begin
    f0:=fu;
    GraFun^[nGraFun]:=fu;
    inc(nGraFun);
   end;
  end;
  DelMatrix(pC1,GA_NP,nVar);
  DelMatrix(pC2,GA_NP,nVar);
  DelMatrix(pD,GA_NP,nVar);
  DelMatrix(pM,GA_NP,nVar);
  DelVector(Fn,GA_NP);
  if fu < f0 then
   vRange:=vRange*sqrt(fu/f0);
end;
```

В последних версиях POC встроен новый алгоритм глобального поиска минимума оценочной функции, **GlobalSearch.**

При обсуждении на Астрофоруме программы POC Владилен Санакоев предложил глобальный поиск новых решений вести по методу «ковровой бомбардировки» и тем самым навёл на мысль об использовании идей планирования эксперимента. Суть этого подхода заключается в методичном поточечном исследовании заданного диапазона варьирования vRange оптимизируемых параметров через определённый шаг. В каждой исследуемой точке по определённому правилу задаются новые параметры ОС и производится одношаговая их оптимизация Симплекс-методом. Затем система переводится в следующую точку, оптимизируется и т.д., пока не будет просканирован весь заданный диапазон vRange пространства параметров.

Из всех исследованных точек за решение принимается точка, в которой оценочная функция достигла самого глубокого минимума.

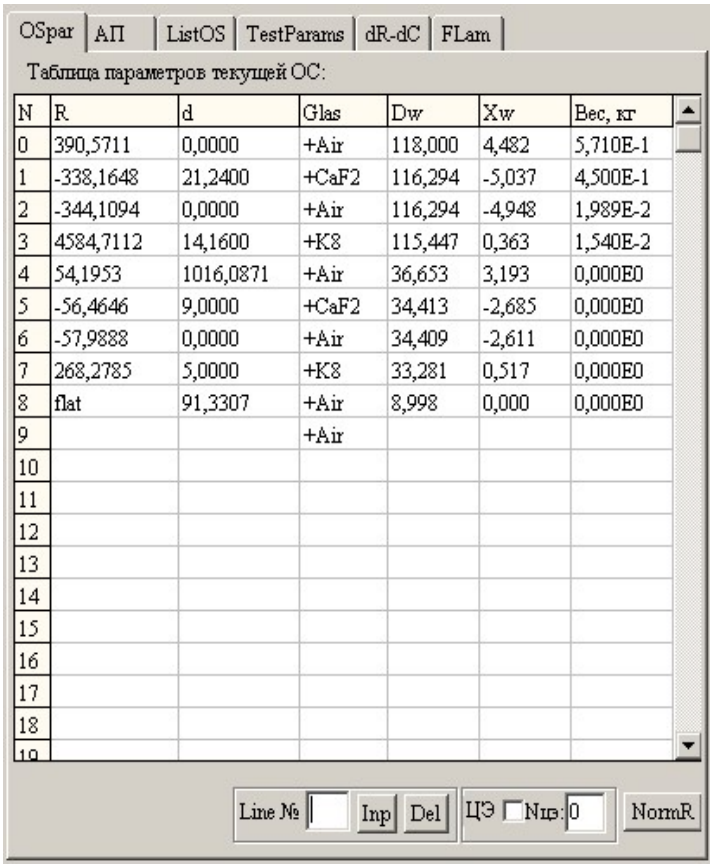
Основным недостатком всех локальных методов оптимизации, в которых вычисляется градиент и которые задействованы в программе РОС, является использование в них одной и той же процедуры LMgold линейного спуска, для которой в процедуре F_r по всем направлениям градиента вычисляется шаг, пропорциональный унифицированному коэффициенту r. С одной стороны, это – срединный, «царский» путь, но, с другой стороны, не пропускает ли такой подход возможных локальных минимумов, лежащих вблизи срединной траектории?

Чтобы повысить вероятность обнаружения таких минимумов в новом алгоритме глобального поиска **GlobalSearch** использован генератор случайных чисел RanMar, обеспечивающий разброс значений переменных оптимизации в пределах половины шага по диапазону варьирования переменных vRange. Остальная работа по поиску «сокрытых минимумов» в окрестности исследуемой точки производится по алгоритму SimplexNM с 6000 рабочих циклов.

```
procedure GlobalSeach(canvas:Tcanvas;Grd0,Grd1:TStringGrid);
var
 i,imn: integer;
 mn: float;

 procedure GScycle(z:float);
 var
  j,k: integer;
  r: float;
 begin
// вводим новые значения переменных оптимизации:
  for j:=0 to nVar do
// случайный разброс - в пределах полушага по диапазону vRange:
   pX^[j]:=oldX^[j]*((RanMar-0.5)/GS_Np+z);
// вычисляем значение целевой функции в плановой точке:
  fu:=sq2ro(pX);
// ищем ближайший минимум:
  Simplex(6000);
  k:=i*2;
  if (z > 0) then
   inc(k);
  if (fu < mn) then
// если минимум ниже глобального, сохраняем его и информируем:
  begin
   imn:=k;
   mn:=fu;
   CopyVector(Xt,pX,0,nVar); // сохранение найденных min-Vars
   Loop(canvas,Grd0,Grd1);   // вывод данных в inf-окно
  end;
// информация о текущем состоянии процесса:
  ClrZon(canvas,15,200,wTestOS-5,215); // очистка старой строки
// информацию о минимуме дополняем информацией о текущем значении:
  canvas.TextOut(20,202,
   'imn='+inttostr(imn)+'. Fgmn='+FormatFloat('0.000',mn*1e3)+
   '. i='+inttostr(k)+', Fgi='+FormatFloat('0.000',fu*1e3)+
   '. Осталось:'+inttostr(GS_Np*2-k)+' циклов, ');
// возвращаем значения параметров ОС в исходной точке:
  CopyVector(pX,oldX,0,nVar);
```

```
 end;

var
 f0,r: float;
begin
// сохраняем исходные данные:
 f0:=fu;
 CopyVector(oldX,pX,0,nVar);
 mn:=f0;
 imn:=GS_Np;
// запускаем цикл обхода пространства варьируемых параметров:
 for i:=1 to GS_Np do
 begin
// очередное смещение по переменным с шагом vRange/GS_Np:
  r:=i*vRange/GS_Np;
// "-" и к толщинам, т.к. в sq2ro d не изменит знак, а обнулится.
  GScycle(-r); // 0 => -vRange
// R также не сколлапсирует, но будет всегда больше полудиаметра
  GScycle(r);   // 0 => +vRange
 end;
 if (mn < f0) then
// сохраняем найденные переменные
 begin
  CopyVector(pX,Xt,0,nVar);
  fu:=sq2ro(pX);
 end
 else
// возвращаемся в исходное положение
 begin
  fu:=f0;
  CopyVector(pX,oldX,0,nVar);
 end;
end;

{=============================================================}
```

9. ВВОД В ПРОГРАММУ НОВОЙ ОПТИЧЕСКОЙ СИСТЕМЫ



N	R	d	Glas	Dw	Xw	Вес, кг
0	390,5711	0,0000	+Air	118,000	4,482	5,710E-1
1	-338,1648	21,2400	+CaF2	116,294	-5,037	4,500E-1
2	-344,1094	0,0000	+Air	116,294	-4,948	1,989E-2
3	4584,7112	14,1600	+K8	115,447	0,363	1,540E-2
4	54,1953	1016,0871	+Air	36,653	3,193	0,000E0
5	-56,4646	9,0000	+CaF2	34,413	-2,685	0,000E0
6	-57,9888	0,0000	+Air	34,409	-2,611	0,000E0
7	268,2785	5,0000	+K8	33,281	0,517	0,000E0
8	flat	91,3307	+Air	8,998	0,000	0,000E0
9			+Air			
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

Слева от таблицы «внешних» параметров, в левом блокноте, расположена вкладка «OSpar».

Вкладка «OSpar».

На этой вкладке находится таблица внутренних параметров рабочей ОС и управляющая панелька «Line №» с двумя кнопками «Inp» и «Del» для добавления/удаления параметров в/из ОС.

В системах с центральным экранированием (ЦЭ) входного зрачка необходимо об этом дать указание программе: поставить флажок в чек-боксе рядом с его названием ЦЭ, а в редакторе Nцэ указать номер поверхности, вызывающей наибольшее экранирование.

Кнопка «NormR» предназначена для приведения номиналов радиусов разработанной ОС в соответствие с каталогом заводских пробных стекол (см. далее раздел «Нормализация радиусов»).

При вводе новой ОС пустые графы таблицы параметров заполняются непосредственными данными. Панелью ввода / удаления строк пользуются для вставки / удаления новой поверхности между уже введенными, т.е. для изменения структуры уже разработанной ОС.

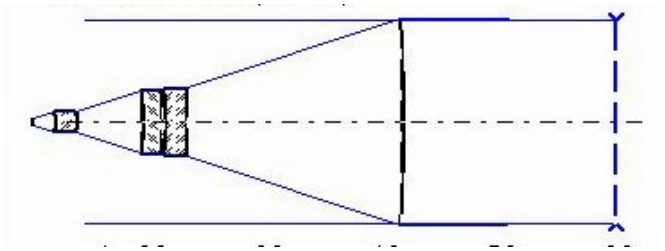
Содержание столбцов таблицы параметров:

R – радиусы кривизны исполнительных (рабочих) поверхностей, в миллиметрах,
d – толщины (или воздушные зазоры между деталями) вдоль оптической оси системы, мм,
Glas – материалы сред (Air – воздух) по каталогам оптических стекол,
Dw – световые диаметры поверхностей с учетом поля 2w (заполняются программно), мм,
Xw - стрелки на Dw (заполняются программно), мм,
Вес, кг – веса деталей (заполняются программно).

Программа РОС располагает обширной базой данных, и достаточно «поднять» соответствующий *.dat-файл, в список которого входит ОС, имеющая требуемую структуру, отмас-

штабировать ее и произвести в ней необходимые изменения. Нередко, тем не менее, приходится заново вводить систему, поэтому рассмотрим этот процесс подробнее.

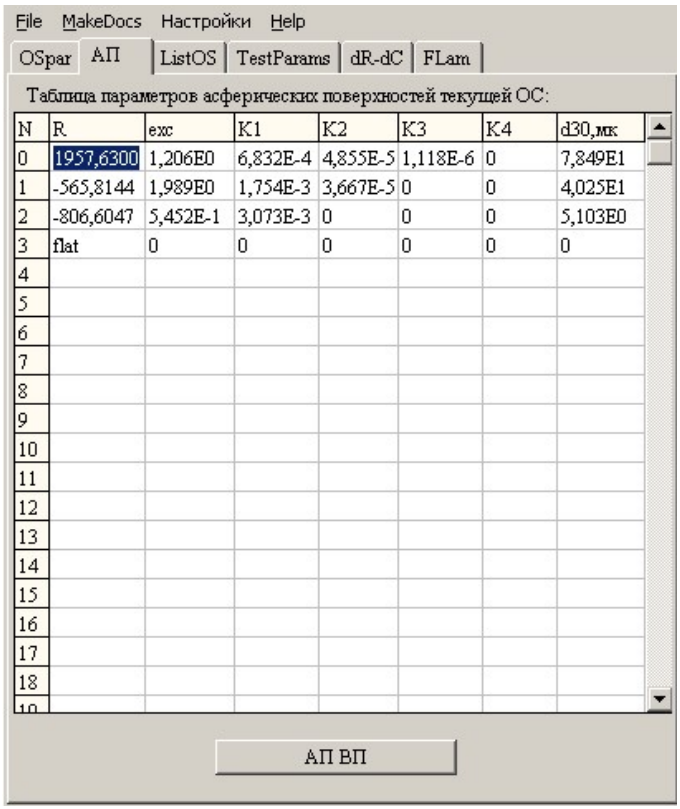
На вкладке **ListOS** программы нажимаем кнопку **NewOS**: программа РОС очищает таблицы «внутренних» и «внешних» параметров и подготавливает свое рабочее пространство к приему новой системы.



ОС с отрицательным значением расстояния от ВЗ до первой поверхности (этот приём использован для совмещения входного зрачка с 2х-линзовым компенсатором).

Начнем с заполнения **Таблицы «внутренних» параметров**.

При вводе данных необходимо иметь в виду следующее:



N	R	exc	K1	K2	K3	K4	d30,мк
0	1957,6300	1,206E0	6,832E-4	4,855E-5	1,118E-6	0	7,849E1
1	-565,8144	1,989E0	1,754E-3	3,667E-5	0	0	4,025E1
2	-806,6047	5,452E-1	3,073E-3	0	0	0	5,103E0
3	flat	0	0	0	0	0	0
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							

- Таблица «внутренних» параметров описывает ОС, включая фокальную поверхность. ОС начинается от входного зрачка (ВЗ), фиктивная плоскость которого на оптической схеме (рис. 3.5) обозначается штриховой жирной линией синего цвета с обратными стрелками на концах. Плоскость ВЗ в ОС не входит – входное окно отнесено к «внешним» параметрам (см. далее), поэтому оно НЕ НУМЕРУЕТСЯ.

Вкладка «АП» (AsSurfs).

В профессиональных программах типа ЗеМакс положению ВЗ отводится особое внимание и его плоскости придан статус дополнительной исполнительной поверхности системы. В РОС применен упрощенный подход, основанный на понимании того, что для хорошо корригированных ОС, как правило, величина аберраций не зависит от положения ВЗ. Тем не менее, в программе предусмотрена возможность смещения положения ВЗ внутрь системы или вовне;

- количество поверхностей в ОС ограничено числом 20. Астрономический телескоп – в отличие от всевозможных «скопов» – является прецизионным инструментом, точность которого зависит от числа и качества изготовления и юстировки его рабочих поверхностей, а, поскольку программа ориентирована на любителя, то откуда и как в любительских условиях возьмутся еще более точные средства для контроля и юстировки многоэлементного прибора? Кроме того, изготовление даже одной точной поверхности – для любителя проблема, а тем более, несколько их десятков;
- нумерация параметров ОС начинается с нуля;
- первая среда заполняет пространство от входного зрачка до первой исполнительной поверхности ОС, и первое расстояние (с номером 0) – это расстояние от ВЗ до первой (номер 0) рабочей поверхности ОС и, если ВЗ расположен справа от первой поверхности (см. рис. 3.4), то это расстояние берется со знаком минус, но знак 0-среды при этом остается положительным.

Для параметра с соответствующим номером N нужно ввести:

R - радиус кривизны исполнительной или – для АП – вершинной сферы,

d - осевое расстояние, начиная с первого (нулевого) расстояния от ВЗ до первой (нулевой) поверхности,

Glas – 6-символьное название материала среды, заполняющей это расстояние, (обозначение материала среды берется из таблицы Glas Catalog, расположенной **на вкладке «GlsCat»**); перед обозначением марки стекла необходимо поставить знак, соответствующий направлению движения света в этой среде: "+" - слева-направо, "-" – справа-налево; например: +Air, +K8, -STK19; при этом в обозначениях марок стекол регистр значим, т.е. «r» и «R» – разные символы, а цифры и знаки пишутся слитно с буквами,

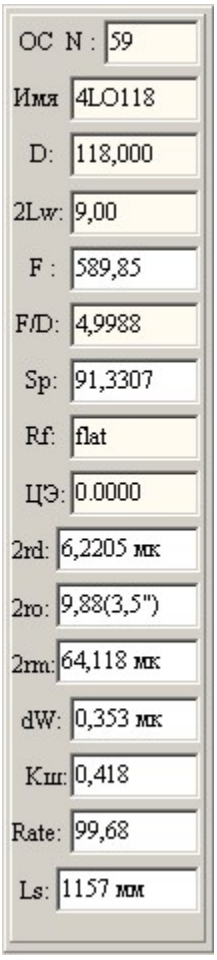


Таблица «внешних» параметров программы РОС.

Если ОС содержит асферические поверхности (АП), то переходим на вкладку «АП» (рис. 3.6) и заполняем соответствующие графы таблицы (столбец радиусов присутствует здесь для справки и заполняется программой автоматически).

exc – вводятся квадраты эксцентриситетов асферических поверхностей. Для планоидной поверхности сюда в миллиметрах вводится значение параметра d30 (вершинной асферичности 3го порядка поверхности);

K3 – K6 – коэффициенты высокого порядка в ряду представления АП в форме:

$$y^2 + z^2 = 2rx + (e^2 - 1) * x^2 + K3 * x^3 + K4 * x^4 + K5 * x^5 + K6 * x^6;$$

и, если в системе нет АП, то эти графы не заполняются (остаются нулевыми).

ВНИМАНИЕ!
В ПРОГРАММЕ РОС НЕ ПРЕДУСМОТРЕНО ОТОБРАЖЕНИЕ ПРОСТРАНСТВЕННЫХ ПОВОРОТОВ ОПТИЧЕСКОЙ ОСИ НА ПЛОСКИХ ЗЕРКАЛАХ ИЛИ В ПРИЗМАХ, А ПРИЗМЫ ДОЛЖНЫ ЗАДАВАТЬСЯ В «РАЗВЕРНУТОМ» ПРЕДСТАВЛЕНИИ – В ВИДЕ ПЛОСКОПАРАЛЛЕЛЬНЫХ ПЛАСТИН.

После заполнения таблицы «внутренних» параметров переходим к заполнению **Таблицы «внешних» параметров**, которая находится в средине Главного окна (рис. 3.7).

После нажатия кнопки «NewOS» программа присваивает новой ОС ее номер (OC N: …) и подготавливает для нее рабочее пространство **в конце** текущего списка ОС.

Вместо вопросов в Таблице внешних параметров вписываются следующие данные:
- Имя – имя новой ОС, состоящее из 12 символов (букв и цифр),
- D – диаметр входного зрачка, мм,
- 2Lw – поле зрения (диагональ матрицы или кадра), мм,
- F/D – фокальный делитель (фокдель),
- Rf – радиус кривизны фокальной поверхности (flat или 9e20 – для плоскости).

Остальные графы заполняются РОС автоматически в процессе работы.

Затем перейдите на вкладку GlsCat, в комбо-боксе «Catalog» из выпавшего списка выберите соответствующий каталог стекла. Затем под меткой Lam, на этой же странице, укажите в микронах крайние значения длин волн (минимальную, а ниже - максимальную) рабочего диапазона спектра, в редакторе nnSpLns укажите число рабочих спектральных линий (максимум – 9, минимум – 2).

На вкладке Тест левого блокнота в панели «Линии спектра» => «Рабочая линия» в окно № введите номер ведущей линии спектра. Например, для видимого диапазона (0.38 – 0.77 мк) при четырех линиях спектра ведущая линия № 1 будет иметь длину волны 0.51 мк.

Во все время заполнения необходимых данных программа РОС терпеливо дожидается основного для нее события – нажатия кнопки “**TestOS**” – нажмите ее.

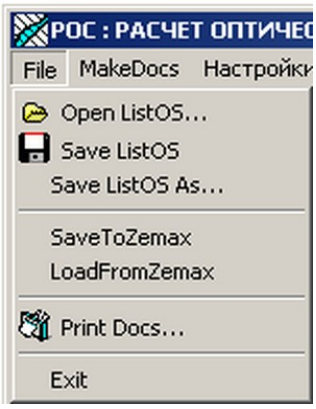
Перед тестированием новой системы POC в том же порядке обходит таблицы с рабочими параметрами ОС и проверяет все данные на корректность и полноту и через оконные сообщения попросит Вас восполнить пропуски и исправить «опечатки». На все вопросы о значении параметра Scale введите единицу (1), а Offset – нули (0). По окончании этого процесса POC произведет автофокусировку системы, установив фокальную плоскость по центральному пучку лучей (если переключатель Sensor select на вкладке Тест установлен в положение Visual), или по крайней полевой точке изображения (если переключатель Sensor установлен в положение Matrix).

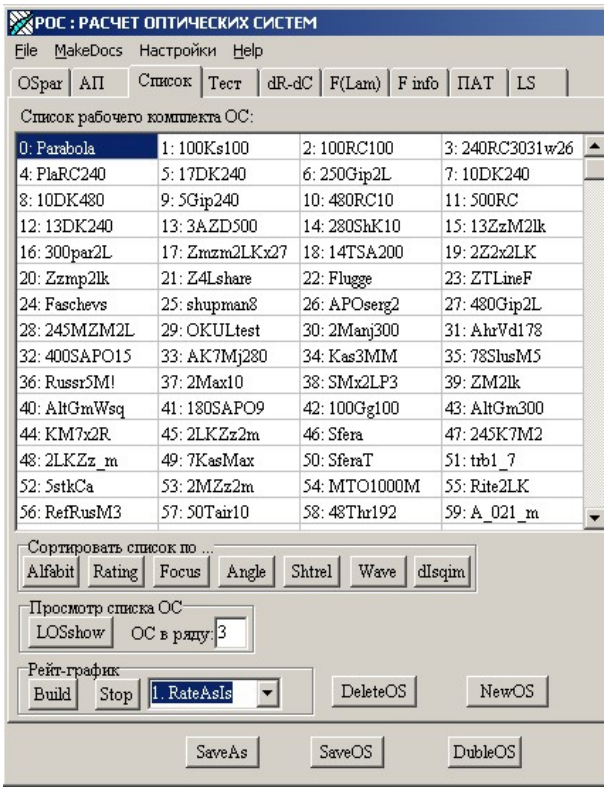
После успешного ввода и тестирования сохраните в памяти ПК новую систему, нажав кнопку «**SaveOS»**. Можно сохранить ОС в другой файл или даже в свой собственный, нажав кнопку «**OSaveAs...**». Затем, войдя в **меню File** программы POC, выберите команду **SaveListOS**: по этой команде сохранится и ОС в файле с рабочим списком, и все рабочие настройки программной среды, включая уставки оптимизатора.

Не забывайте сохранять свои изменения: эту инициативу POC оставляет за Вами.

10. ИСПОЛЬЗОВАНИЕ ОС ИЗ БАЗЫ ДАННЫХ РОС

При нажатии пункта меню File из строки меню выпадает список команд, в котором нужно выбрать команду Open ListOS…, а в открывшемся системном окне «Открыть…» найти папку …ROS\Data, в которой выбрать файл с расширением *.dat, содержащий список искомых ОС. После загрузки этого файла необходимо открыть вкладку «Список» левого блокнота программы РОС и из него выбрать ОС, наиболее близко совпадающую по структуре с той ОС, которую необходимо разработать.





Вкладка «Список» левого блокнота программы РОС

Таблица со списком рабочего комплекта оптических систем размещается на вкладке ListOS левого блокнота.

Для выбора ОС из рабочего списка необходимо на вкладке **«ListOS»** щелкнуть левой кнопкой мышки по не подсвеченной ячейке таблицы со списком ОС. При этом выбранная система загрузится в рабочую область программы, и ее оптическая схема появится в Тест-окне.

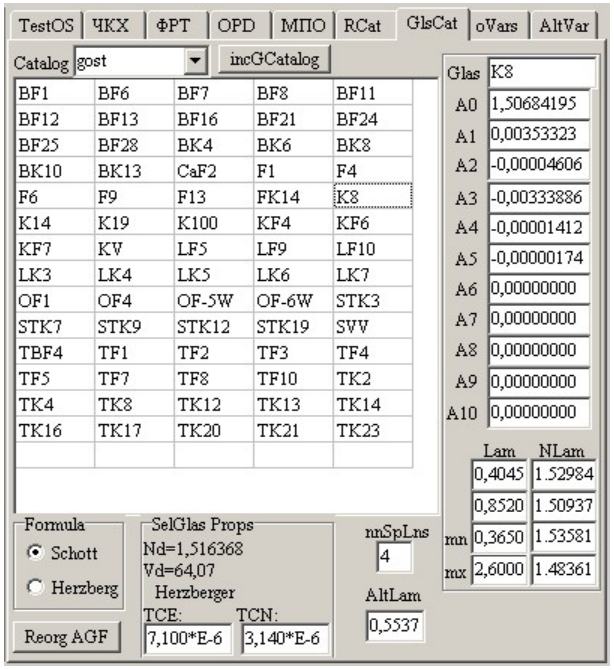
Здесь же, на вкладке **«ListOS»,** расположены следующие кнопки:

- для удаления ОС из списка (**DeleteOS**),
- для подготовки программы к вводу новой ОС (**NewOS)**,
- управляющие элементы
  = для сортировки списка ОС (кнопки Alfabit, Rating, Focus, Angle, Strel, Wave, dIsqim),
  = просмотра по нескольку ОС в одном окне (ListOSshow),
  = построения графика рейтингов (Build).

11. УСТАНОВКА КАТАЛОГА СТЁКОЛ

На странице GlsCat правого блокнота пользователь может произвести необходимые манипуляции с каталогами стекол, параметрами стекол и выбрать или установить рабочие линии спектра.

Комбо-бокс Catalog позволяет выбрать один из 11 каталогов стекол по их сокращениям:



Вкладка GlsCat правого блокнота программы .

'cornng', – каталог стекол зарубежной фирмы «Корнинг»;
'gost', – каталог отечественных стекол, соответствующих требованиям ГОСТ;
'lzosm', – последний каталог стекол ЛЗОС;
'hikari', – каталог стекол зарубежной фирмы «Хикари»;
'hoya', – каталог стекол зарубежной фирмы «Хойя»;
'lzos_5', – вариант каталога стекол завода ЛЗОС;
'lzos_w', – вариант каталога стекол завода ЛЗОС с префиксом W;
'ohara', – каталог стекол зарубежной фирмы «Охара»;
'opal', – каталог отечественных стекол, взятый из программы ОПАЛ;
'schott', – каталог стекол зарубежной фирмы «Шотт»;
'sumita' – каталог стекол зарубежной фирмы «Сумита»;

Список каталогов стекол.

Радио-бокс «Formula», расположенный под таблицей стекол текущего каталога, позволяет выбрать, по какой формуле будут рассчитываться показатели преломления рабочих стекол системы. Эта опция задействуется после нажатия кнопки Reorg AGF: программа производит изменения в базе данных рабочего каталога, которые на диске не сохраняются, но используются только во время работы с ОС.

По нажатии кнопки incGCatalog открывается системное окно «Открыть» и после выбора пользователем нового каталога стекол программа РОС обрабатывает указанный AGF-файл, адаптирует его для своей работы и включает в свой список каталогов стекол.

Правая половина вкладки GlsCat используется для просмотра параметров конкретного стекла из текущего каталога, а также для установки для Вашей ОС рабочих линий спектра:

Glass – название стекла из текущего каталога,

Nd – каталожное значение показателя преломления для d-линии спектра,

Vd – каталожное значение числа Аббе для d-линии спектра,

Herzberger – название формулы, принятой в текущем каталоге для расчета показателей преломления рабочих стекол во время работы программы,

TCE и TCN – каталожные значения коэффициентов теплового расширения и изменения показателя преломления для выбранного стекла,

A0…A10 – значения коэффициентов в аппроксимирующей формуле для выбранного стекла, взятые из текущего каталога,

Lam и NLam – две крайние длины волн (минимальная и ниже – максимальная), определяющие рабочий диапазон спектра и устанавливаемые пользователем, и значения показателей преломления на этих длинах волн, заполняемые программой,

mn, mx – минимальная и максимальная длины волн рабочего диапазона для данной ОС, определенные программой по параметрам стекол, из которых составлена данная ОС,

nnSpLns – окно редактора, в котором пользователь устанавливает число линий спектра, по которым программа будет тестировать и оптимизировать данную ОС.

AltLam – информационное окно, в которое программа выводит во время своей работы значение длины волны, на которой производится тестирование ОС; номер тест-линии устанавливается пользователем в панели spLnSelect на вкладке Тест (см. раздел «Тестирование ОС»).

12. АСФЕРИЧЕСКИЕ ПОВЕРХНОСТИ В ОС

Программа РОС «понимает» три типа поверхностей:

1. АП 2-го порядка – сфера (в т.ч. плоскость), эллипсоид, параболоид и гиперболоид. При этом знак у квадратов эксцентриситетов для всех этих АП – положительный (в отличие от программы ZeMax). Для сплюснутого сфероида знак $e^2 < 0$ – отрицательный.

2. Планоидные поверхности типа профиля Кербера.

3. АП высокого порядка – ретушированные поверхности 2го порядка или планоидные поверхности (типа пластин Б. Шмидта) с составляющими высокого порядка – обобщенный профиль Кербера.

Профиль Кербера 3-го порядка описывается уравнением

$$x = \delta_0*(m*\rho^2 - \rho^4),$$

где x - текущая абсцисса поверхности или «стрелка» профиля, $\rho = y/Hs$ - относительная ордината на световом окне поверхности, y – текущая ордината поверхности, Hs – полудиаметр поверхности (предельное значение ординаты Y на данной поверхности); m - параметр профиля, определяющий относительно оси положение точки перегиба профиля. При m = 2 для y = Hs: $x = \delta_0$. Параметр δ_0 характеризует численное отступление профиля от плоскости, касательной в его вершине – при параметре m = 0 и при m = 2, численно $\delta_0 = x(Hs)$.

Если АП представлена в полярных координатах, то профиль Кербера позволяет в едином ключе рассматривать все АП, которые предстают как одна АП с различными значениями вершинной асферичности: начинается со сплюснутого сфероида, затем – сфера, в конце этого ряда – гиперболоид. АП ВП удобно анализировать с помощью обобщенного профиля Кербера, определяя целесообразность усложнения вводимой асферической поверхности. Более подробно с описанным представлением АП можно познакомиться в курсе лекций «Основы формообразования оптических поверхностей», В.И. Каширин, Екатеринбург, УПИ, 2006.

Окно просмотра параметров АП

При нажатии на вкладке «AsSurfs» **кнопки «ViewHiLevelAsfSurfaceis»** открывается окно "Параметры АП ВП", в котором представлены результаты анализа структуры асферичности исполнительных поверхностей ОС относительно профилей Кербера различного порядка:

Окно анализа структуры асферичности АП.

В **таблице "Параметры АП"** приведены:
D - световой диаметр АП, мм,
Ro - радиус кривизны вершинной сферы, мм,
Rбсс - радиус кривизны ближайшей (трех-точечной) сферы сравнения АП, мм,
exc - квадрат эксцентриситета поверхности,
d03 – вершинная асферичность 3-го порядка, мк,
d05 – вершинная асферичность 5-го порядка, мк,
sqv - остаточная среднеквадратичная ошибка АП, нм.

Таблица сопоставления АП с профилями Кербера представляет:
ro = y / H - относительная ордината на АП,
Хбсс - абсцисса на ближайшей сфере сравнения,
ХАП - абсцисса на АП,
dX = Хбсс - ХАП - разность абсцисс,
dt - пересчет разности по нормали к ближайшей сфере сравнения,
kerb3 - абсцисса на профиле Кербера 3-го порядка,
dt-kerb3 - разность между нормализованной асферичностью и профилем Кербера 3-го порядка,
kerb5 - профиль Кербера 5-го порядка,
ddt - остаточная ошибка АП.

Для указания программе POC о том, что поверхность стала асферической необходимо:
1. В панели «exc» вкладки optVars в строке с порядковым номером асферизуемой поверхности ввести символ «v» в столбце Var; если АП луч проходит дважды, то в столбце «pic up» ввести номер поверхности, которую луч пересекает первой, а в столбце Scale указать коэффициент трансформации активной асферичности в пассивную.
2. Перейти на **вкладку «AsSurfs»**.
3. В **графе exc** ввести значение квадрата эксцентриситета АП. Для планоидной поверхности здесь вводится "вершинная асферичность" d30, то есть максимальное отступление планоида от плоскости, касающейся его в вершине, в мм. Закон изменения профиля АП определяется профилем Кербера 3-го порядка.
4. Для АП высокого порядка ввести **коэффициенты высокого порядка К3, К4, К5 и К6**, определяемые из уравнения АП:

$$y^2 + z^2 = 2rx + (e^2 - 1) * x^2 + K3 * x^3 + K4 * x^4 + K5 * x^5 + K6 * x^6.$$

Рекомендуется установить следующие предварительные значения коэффициентов:
К3 = 1e-4, К4 = 1e-6, К5 = 1e-8, К6 = 1e-10.
5. Вернуться на вкладку **«МПО»** и откорректировать уставки МПО. Для успешной работы с АП ВП необходимо на панелях «Цель МПО», «Уменьшить» включить одновременно обе опции «2ro» и «Wскв».
6. Нажать кнопку «**TestOS**» для принятия произведенных изменений.

7. Запустить **МПО**, нажав **кнопку «Старт МПО».**

13. ТЕСТИРОВАНИЕ ОПТИЧЕСКОЙ СИСТЕМЫ

Программа РОС оснащена большим набором средств для специального тестирования оптических систем. Назначение элементов вкладки «Тест» левого блокнота (Рис. 6.1).

1. Радио-бокс Lw. В положении «0» тестируется центральная точка изображения, в положении Lw – точка на краю поля зрения.
2. На панели «Линии спектра» можно выбрать из рабочего спектрального диапазона номер рабочей спектральной линии, на которой программа будет производить тестирование ОС; при выборе крайних линий, ОС тестируется на минимальной или максимальной длине волны (см. раздел «Работа с каталогами оптических стекол»).
3. Бокс «Зональная ошибка» предназначен для оценки влияния осесимметричных кольцевых ошибок на качество изображения, даваемого разрабатываемой ОС:
 - число волн N зональной ошибки, имеющей синусоидальную форму, которые могут уложиться на диаметре входного зрачка. По умолчании устанавливается число 3.333, как наиболее действенное;
 - амплитуда Wmax зональной ошибки в микронах. При Wmax = 0 эта опция игнорируется.
4. Волновая ошибка ОС может отображаться, как в кольцах Ньютона, так и в полосах (при установке флажка «Показать» на панели «Интерферограмма»); число полос устанавливается в окне «N».
5. Радио-бокс «Сенсор»: при выборе Visual фокусировка производится на плоскость Гаусса по центральной точке изображения, а при Matrix - фокусировка производится по минимуму среднеквадратичного значения пятна рассеяния, осредненного по спектру и полю. Разные системы показывают лучшие результаты при установке только на «свой» тип сенсора.
6. Панель «TstObjct» предназначена для выбора объекта для тестирования: по 1й и 2м искусственным звездам, по 2м щелям, а также по радиальной или 3х/4х-штриховой мире. В этой панели можно изменить шаг между элементами изображения – угловое расстояние в секундах между звездами/штрихами; для справки здесь приведен угловой размер дифракционного кружка Эйри (2rd) для ОС, находящейся в данный момент в работе.

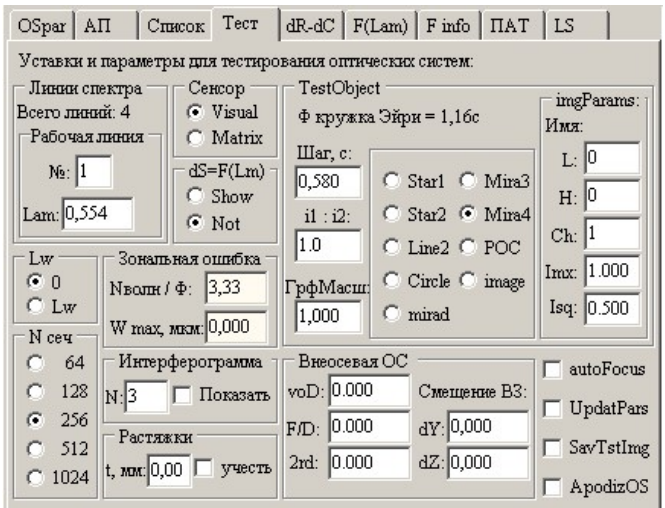


Рис. 6.1. Страница «Тест» с уставками для тестирования оптических систем.

В окна ImgParams выводятся параметры цифрового тест-изображения, загруженного из файла: L – ширина кадра в пикселях, H – высота кадра в пикселях, Isq – среднеквадратичная яркость пикселя в изображении, Imx - максимальное значение яркости пикселя, Ch – цветность изображения (1 или 3 канала).
7. Редактор «ГрфМасш» предназначен для изменения масштаба у графика интенсивностей в поперечном сечении тест-изображения.
8. Панель dS = F(Lam) предназначена для включения или отключения опции построения зависимости смещения фокальной плоскости от длины волны в рабочем диапазоне спектра.
9. В панели «N сеч» можно выбрать число сечений входного зрачка, одинаковых для меридионального и сагиттального направлений, которые РОС использует в своих вычислениях ФРТ, ЧКХ и др. По требованию процедуры БПФ это число производно от двойки: 64, 128 и т.д.; при N, большем 1024, время выполнения тест-процедур становится неприемлемо большим.
10. Панель «Растяжки» предназначена для изменения в редакторе «t» значения толщины растяжек в Вашем рефлекторе. При установке флажка «Учесть» будет производиться тестирование и расчет критериев качества системы с учетом дифракции на растяжках.
11. Панель «Внеосевая ОС» заполняется в том случае, если ОС внеосевая:
- dY – смещение центра входного зрачка (ВЗ) поперёк оптической оси в меридиональной плоскости;
- dZ – смещение центра ВЗ поперёк оптической оси в сагиттальной плоскости.
- voD – диаметр внеосевого ВЗ.
Редакционные окна F/D и 2rd заполняются автоматически.
12. При установке флажка autoFocus программа перед тестированием производит подфокусировку ОС по минимуму среднеквадратичного значения пятна рассеяния, осредненного по спектру и полю зрения; при снятом флажке автофокусировка не производится.
13. Чекбокс UpdateMast необходимо включать для масштабирования параметров текущей ОС (при изменении её диаметра или фокделя).
14. При установке флажка SaveTestImg пользователю будет предлагожена возможность сохранить дифракционное изображение тест-объекта, построенное тестируемой системой.
15. При установке флажка в чек-боксе Apodization волновая ошибка системы обнуляется по всему входному зрачку и её тестирование производится так, как если бы она была идеальной. Практически этого можно достичь путем полной асферизации или модификации пропускания.

2d- и 3d-графики частотно-контрастных характеристик

На второй вкладке правого блокнота (ЧКХ) находятся две кнопки: 2d_ ЧКХ и 3d_ЧКХ:
при нажатии первой выдается график ЧКХ в меридиональном сечении системы, при нажатии второй – псевдо-объемное изображение ЧКХ в координатах меридионального и сагиттального сечений.

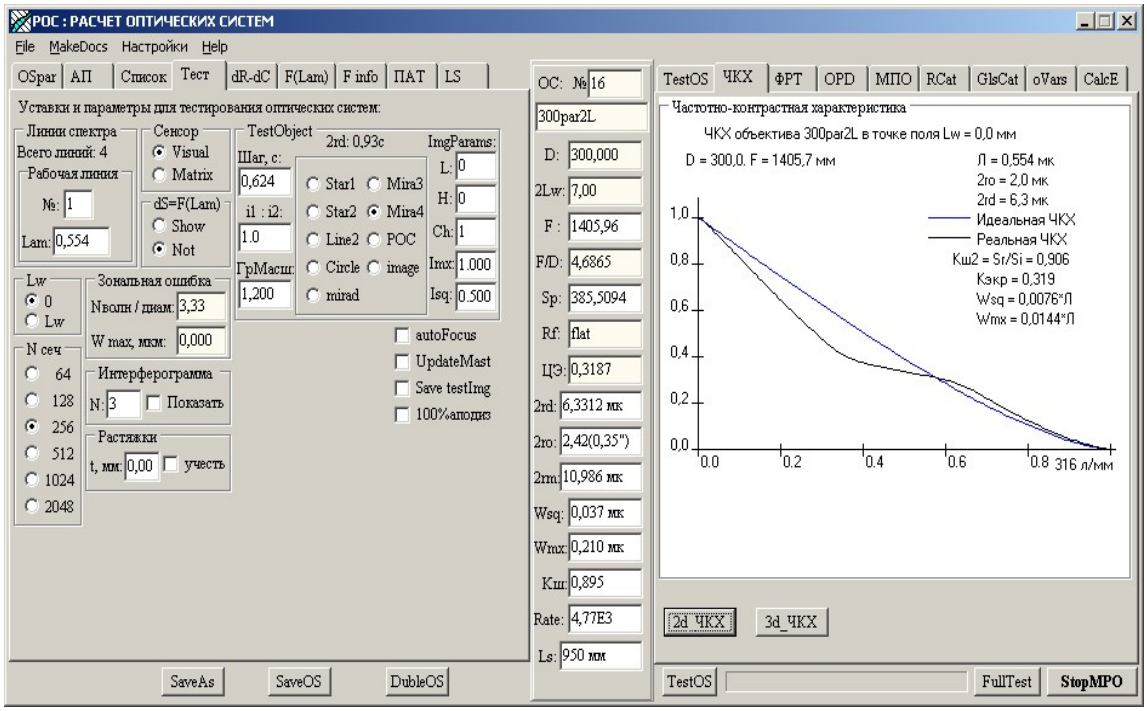


Рис. 6.2. Программа РОС при тестировании ОС по ЧКХ.

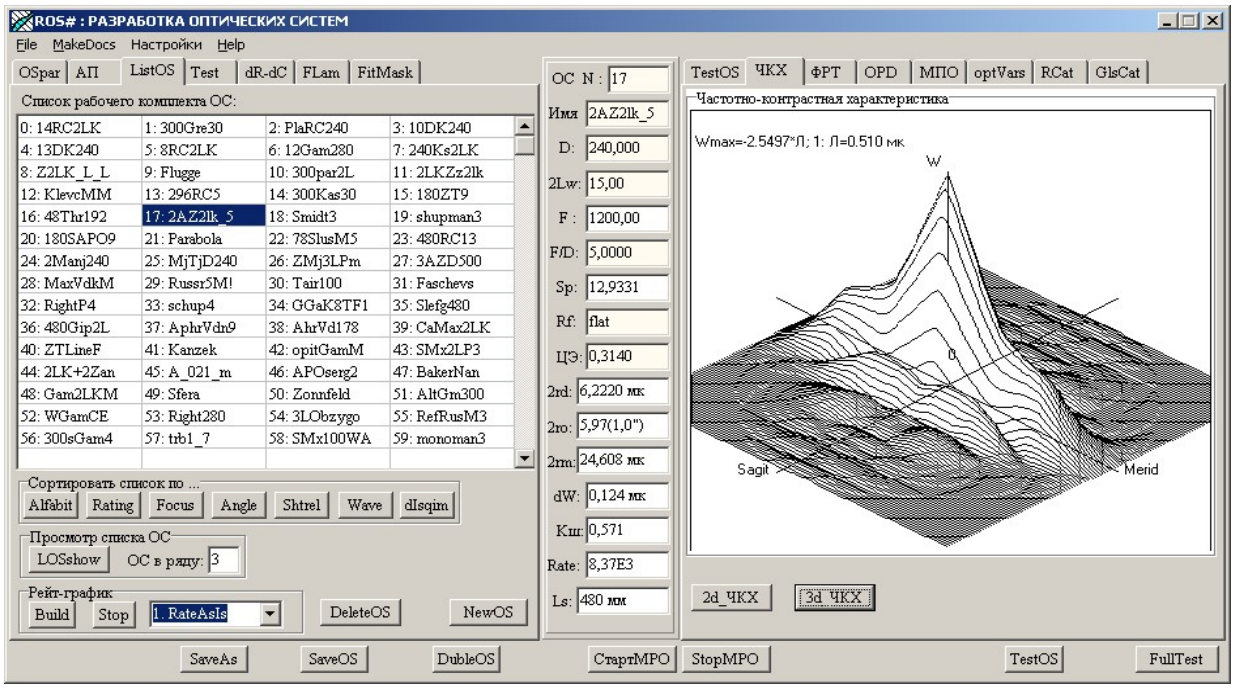


Рис. 6.3. 3d-график ЧКХ.

Функция рассеяния точки (PSF) и дифракционные тесты по синтезированным звездам, щелям и мИрам

На третьей вкладке правого блокнота (ФРТ) имеются две кнопки: ФРТ и Image-TEST. При нажатии первой выдается объемный график функции рассеяния точки, при нажатии второй рисуется дифракционная картина изображения тест-объекта. На панели TestObjct вкладки «Тест» левого блокнота можно выбрать тип тест-объекта, установить угловое расстояние между (Шаг) звездами/пикселями/штрихами по угловому размеру дифракционного кружка Эйри (2rd); для 2х звёзд установить соотношение их яркостей (i1 : i2) и указать масштаб (ГрМасш) для графиков распределения яркости (интенсивности) в сечениях дифракционных картин.

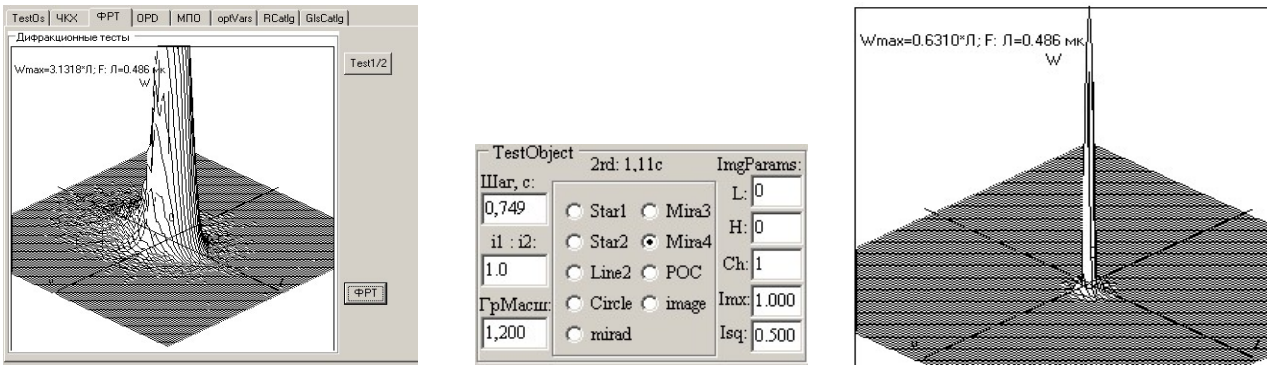


Рис. 6.4. ФРТ на краю поля (слева) и в центре (справа);
в середине – панель TestObjct на вкладке «Тест».

Контраст между звездами и штрихами можно определить по сечениям на дифракционных картинах, направление обхода которых обозначено пунктирными линиями. Приведенное на графиках значение контраста S/N(0,0) определено для центра дифракционной картины.

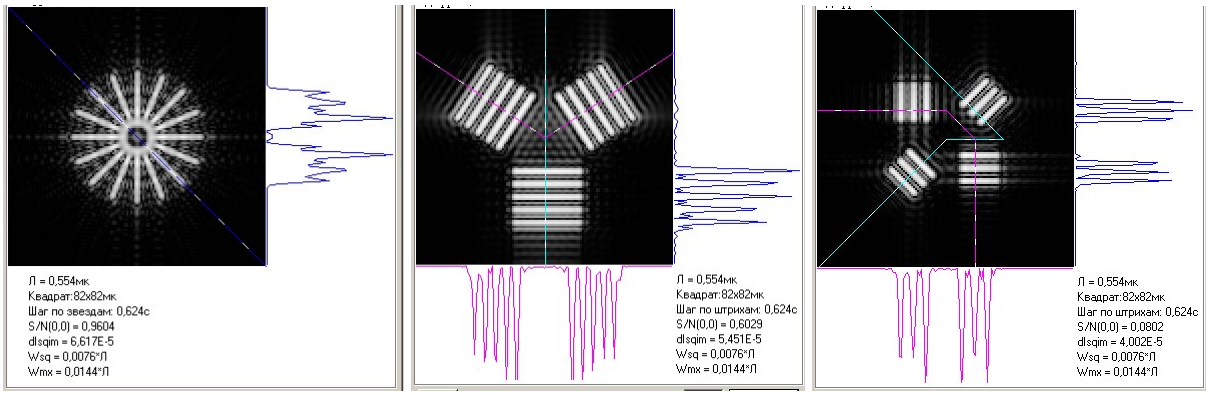
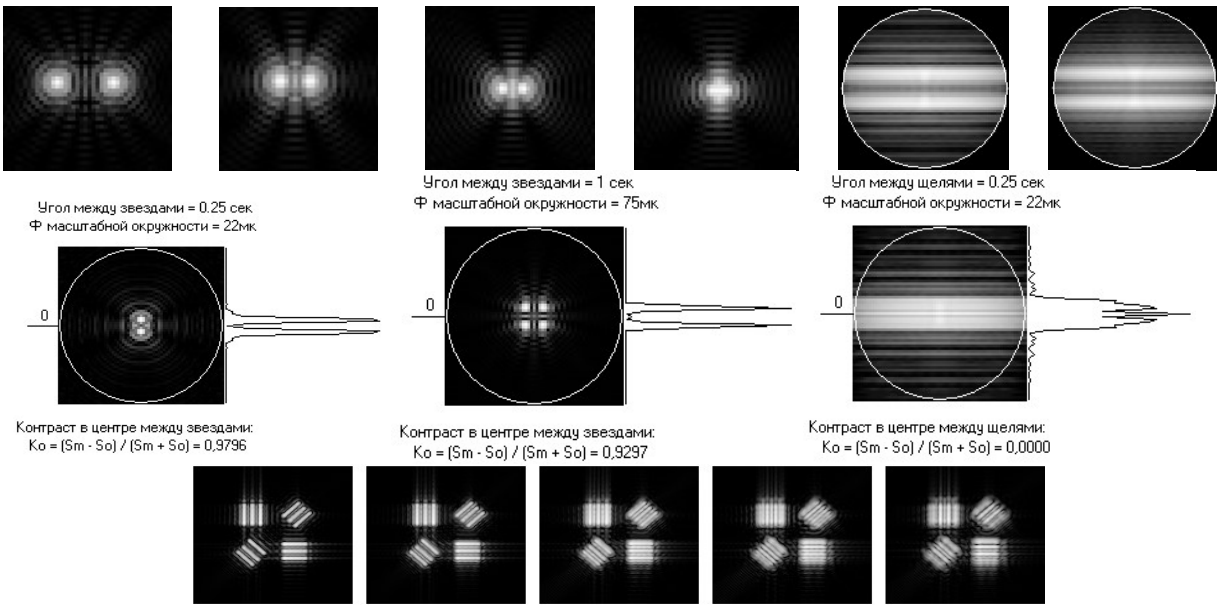


Рис. 6.5. Направление обхода тест-изображения миры при построении графика интенсивности.

На рисунке 6.6 представлено несколько дифракционных изображений тест-объектов.



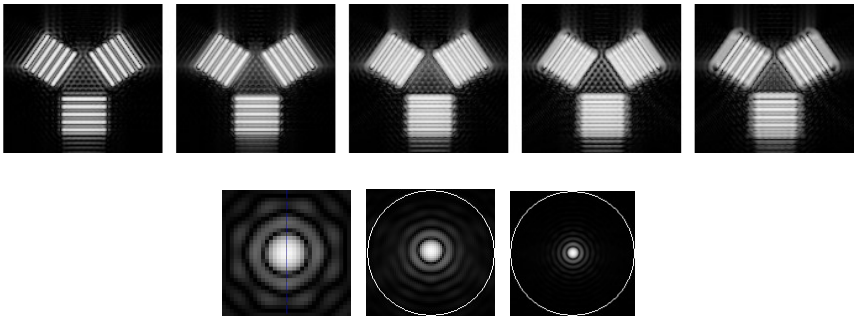


Рис. 6.6. Дифракционные изображения звезд, щелей и элементов штриховых мир.

В нижнем ряду на рисунке 6.6 представлено дифракционное изображение искусственной звезды при разных увеличениях (значениях коэффициента difrImgMast, устанавливаемое на вкладке «ФРТ»). Надо иметь в виду, что при difrImgMast > 2 происходит отсечение краёв у зрачковой функции, что приводит к искажению результатов преобразований процедурой БПФ и образованию артефактов на дифракционных изображениях в виде прерывистости и некруглости дифракционных колец и ядер.

2х и 3х-мерные графики OPD и волновая ошибка в кольцах Ньютона

На 4й вкладке правого блокнота находятся три кнопки «Кольца Ньютона», View_W и Interference, а также панелька GrfDim для выбора размерности графика.

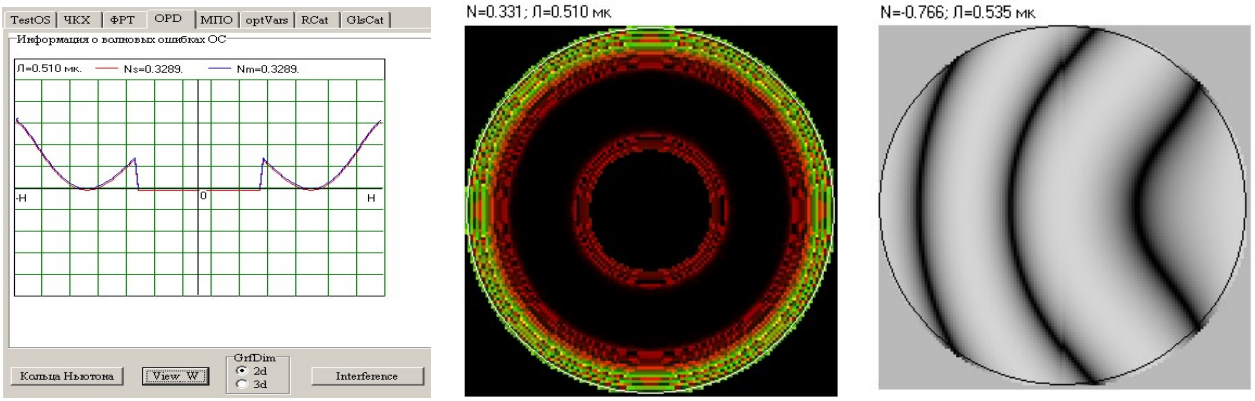


Рис. 6.7. Вкладка OPD (Optical Path Difference) и интерференция в кольцах Ньютона и в полосах.

При нажатии кнопок «Кольца Ньютона» или Interference выдаются изображения колец Ньютона или полос, которые можно было бы увидеть в воздушном зазоре между волновым фронтом, вышедшим из оптической системы, и ближайшей сферой сравнения к нему.

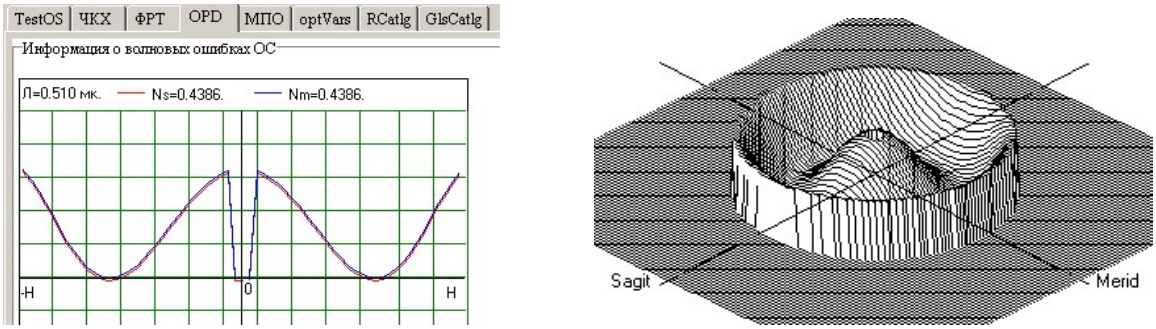
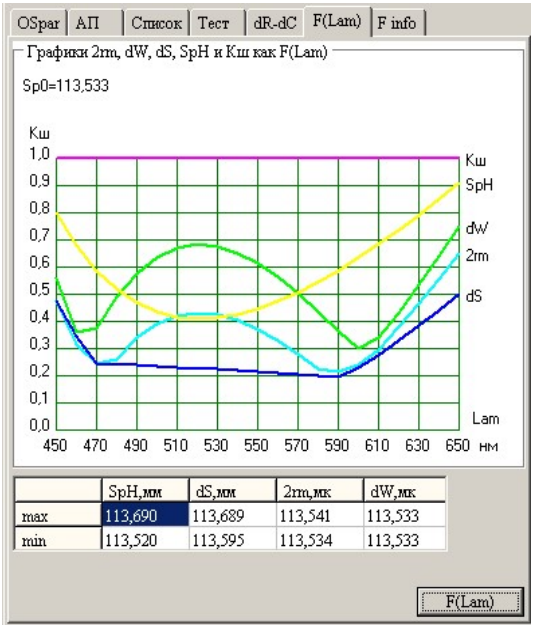


Рис. 6.8. Слева – волновая ошибка в меридиональном и сагиттальном сечениях; справа – ее 3d-график.

При нажатии кнопки View_W выводятся 2х или 3х-мерные графики волновой ошибки системы (зрачковой функции, Pupil Function). Необходимо отметить однозначную корреляцию волновой ошибки dW с величиной среднеквадратичной разности яркостей на идеальном и реальном изображениях (dIsqim): чем меньше у ОС волновая ошибка dW, тем меньше dIsqim. Такой же взаимосвязи, например, между ЧКХ и dIsqim, не наблюдается.

Зависимость критериев качества оптической системы от длины волны

На вкладке F(Lam) представлены кривые зависимости от длины волны коэффициента Штреля, Кш, и среднеквадратичных значений размера наибольшего пятна рассеяния 2rm, волновой ошибки dW, значений последнего отрезка SpH для края входного зрачка и размаха продольной ошибки dS тестируемой системы.



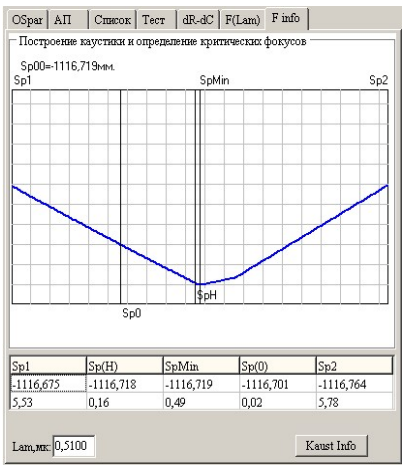
. Sp0 – исходное значение последнего отрезка.

В Таблице под графиком указаны максимальные и минимальные значения, по которым можно определить масштаб для соответствующего параметра. Градуировка ординаты от 0 до 1 произведена для Кш.

Рис. 6.9. Зависимость параметров ОС от длины волны.

Построение графиков производится при открытии вкладки Flam левого блокнота или по нажатии кнопки F(Lam).

Построение каустики и определение положения фокальной «перетяжки»



Тестирование ОС или ее элементов в любительской практике обычно производится теневым методом Фуко. В этом случае полезно иметь сведения о положении как параксиального (Гауссового), так и краевого фокусов.

Известны трудности точного определения параксиального фокуса. Поэтому сначала определяется краевой фокус, для чего необходимо перекрыть входной зрачок ОС центральным

экраном (ЦЭ), оставив на краю входного зрачка узкий кольцевой зазор.

Рис. 6.10. График каустики для ОС Райта Ф280/4.

Затем относительно полученного краевого фокуса определяются положения остальных фокусов: минимального пятна и краевого для отверстия в главном зеркале. Гауссов фокус целесообразнее вычислять теоретически.

Особый интерес представляет положение минимального пятна, совпадающее с положением «перетяжки» на каустике – минимума на кривой, огибающей пучок лучей, сходящихся к фокусу системы.

На вкладке «F info» левого блокнота строится график каустики для выбранной спектральной линии, в таблицу заносятся расчетные данные:

- Sp1 и Sp2 – определяют диапазон последних отрезков; под ними – значения полудиаметров каустики в микрометрах;
- Sp(H) и Sp(0) – последние отрезки для края и центра входного зрачка; под ними – величина ошибки определения размера этих пятен, обусловленная дискретностью шага по оси абсцисс;
- SpMin – положение «перетяжки» на каустике относительно последней поверхности ОС; под этим значением выведен минимальный полудиаметр каустики в микрометрах;
- Sp00 – исходное значение последнего отрезка, установленного программой по минимуму суммарного пятна рассеяния.

Полное тестирование оптической системы

При нажатии в Главном окне кнопки «FullTestOS» открывается новое окно «Полный тест ОС …», в котором расположена кнопка «Старт»: по ее нажатии производится полный тест, по результатам которого можно проследить за динамикой роста аберраций в зависимости от размера поля зрения системы.

В левом верхнем углу окна «Полный тест» приводится копия обычного тест-окна программы с графиком продольных ошибок dS = F(Lam), справа от которой расположена сокращенная Таблица с параметрами ОС. В этой Таблице указано число сечений Nsec входного зрачка, использованное программой при анализе ОС, а также значение последнего отрезка optSp, оптимизированного на минимум волновой ошибки. Знаком «*» отмечаются асферические поверхности.

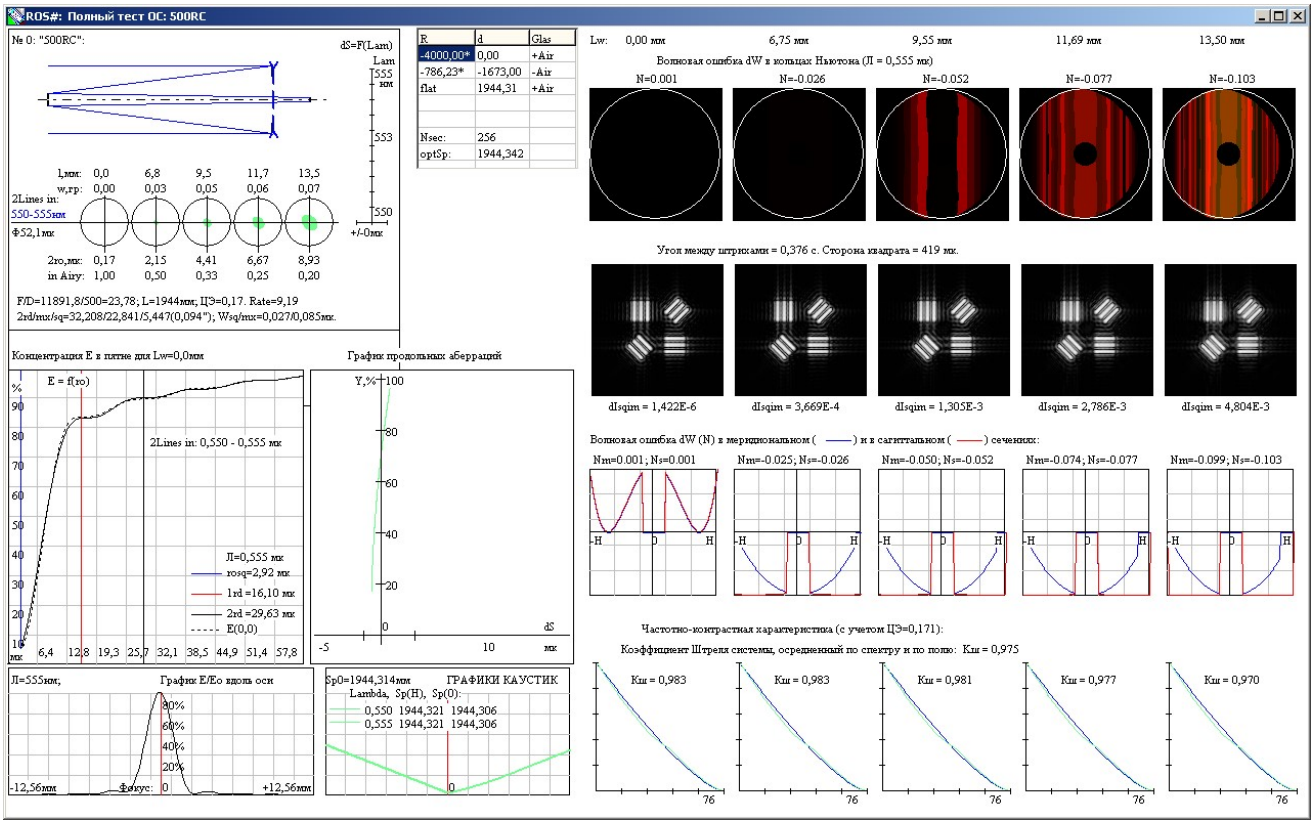


Рис. 6.11. Окно с информацией о полном тестировании системы 500RC Ф500/23.78.
В качестве тест-объекта использована синтезированная 4х штриховая мира.

Под копией тест-окна располагается график концентрации излучения в пятне рассеяния, построенный для заданной пользователем полевой точки (0 или Lw) и для линии спектра, выбранной на вкладке Тест. Штриховой линией построен график концентрации энергии в пятне для ОС, имеющей нулевое значение волновой ошибки, по формуле

$$E = 1 - J_1^2(ro),$$

где J_1 – функция Бесселя первого рода первого порядка.

Вертикальными прямыми на графике E = f(ro) для ведущей линии спектра отмечены значения первого (красный цвет) и второго (черный) дифракционного минимумов, а также среднеквадратическое значение половины диаметра пятна рассеяния, rosq (синий).

Графики продольных ошибок dS = F(y) вычисляются для меридионального сечения ОС для всех рабочих линий спектра.

Еще ниже приведен график расфокусировок (изменения интенсивности сфокусированного системой излучения вдоль оптической оси) для ведущей линии спектра, а также графики каустик.

В правой части окна полного тестирования ОС расположены картины колец Ньютона (или интерференционных полос) и дифракционных изображений тест-объекта, графики волновой ошибки в меридиональном и сагиттальном сечениях, а также графики ЧКХ – в зависимости от линейного размера поля зрения. Здесь же указано значение коэффициента Штреля, осредненное по полю и спектру.
Графики концентрации энергии, ЧКХ и каустик приведены для всех рабочих линий спектра.

При разрешении дисплея менее 1024x768 полное тестирование не производится.

Тестирование ОС по всем тест-объектам в одном окне

При нажатии в Главном окне кнопки «AllObjTest» открывается новое окно «Тест по всем объектам ОС: …», в котором расположена кнопка «StartTest»: по ее нажатии производится тестирование ОС с использованием семи синтезированных тест-объектов по всему полю зрения, начиная от его центра, к краю. Для остановки процесса тестирования нажмите любую клавишу, после чего закройте Тест-окно.

По результатам тестирования можно проследить за качеством изображения в зависимости от размера поля зрения системы и типа тест-объекта.

Первые четыре тест-объекта (одна искусственная звезда, две ИЗ, две линии и кольцо) представлены в 2х-кратном увеличении, остальные три (радиальная мира, 3х- и 4х-штриховая миры) изображаются в масштабе 1 : 1. Над изображением каждого тест-объекта в относительных единицах указан уровень фона в нём.

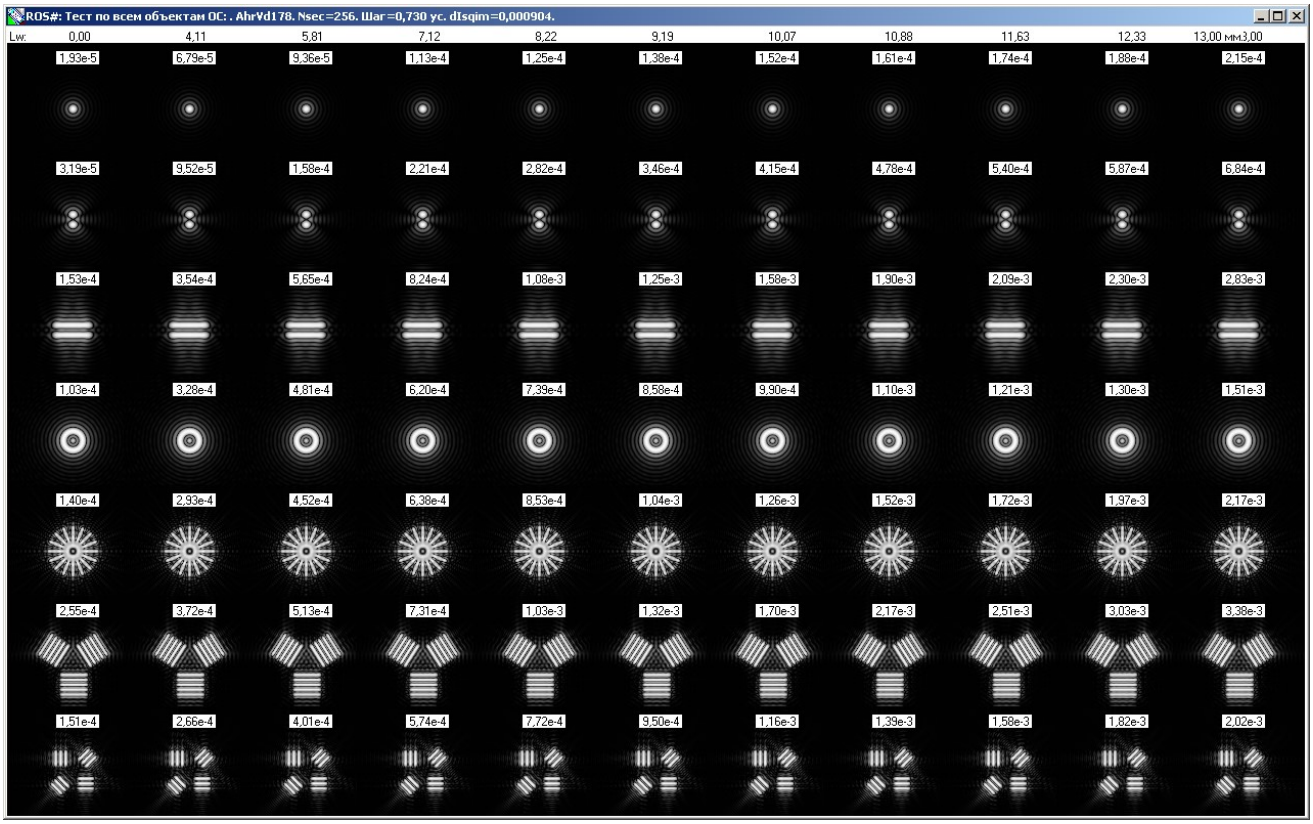


Рис. 6.12. Тестирование 60-мм поля изображения системы AhrVd178 по семи тест-объектам.

В строке названия Тест-окна приведены следующие данные: Nsec – число сечений входного зрачка, использованное при вычислении зрачковой функции; «Шаг» - расстояние между искусственными звездами в тест-объектах, в угловых секундах; dIsqim – среднеарифметическое значение уровня фона, в %, определённое по всем изображениям.

Выбором числа сечений на вкладке «Тест» левого блокнота программы можно обеспечить необходимый масштаб изображения тест-объектов.

При разрешении дисплея менее 1024x768 тестирование по всем тест-объектам не производится.

Тест по снимкам реальных объектов

На странице «Тест» левого блокнота в боксе TstObjct выберите image (рис. 6.13), перейдите на страницу ФРТ правого блокнота и нажмите кнопку Image-TEST – откроется системное окно «Открыть». В окне «Открыть» в папке ros\docs\imgs\ выберите нужный файл с расширением *.fit для тестирование своей системы по изображению реального объекта.

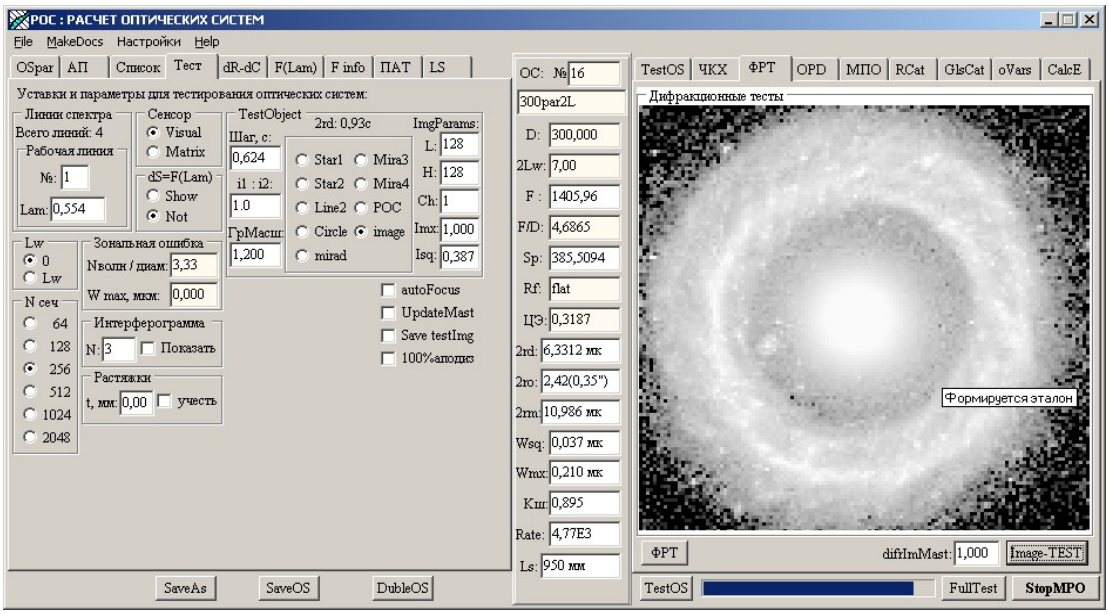


Рис. 6.13. РОС после вывода на вкладку ФРТ изображения выбранного тест-объекта.

РОС приступит к выполнению первого этапа тестирования – формированию эталонной зрачковой функции по изображению тест-объекта, которую имел бы Ваш объектив, не будь у него аберраций. На рисунке 6.13 программа сигнализирует о ходе обработки тест-изображения: заполняет окно прогресс-бара и выводит на изображение взятое в рамку сообщение «Формируется эталон».

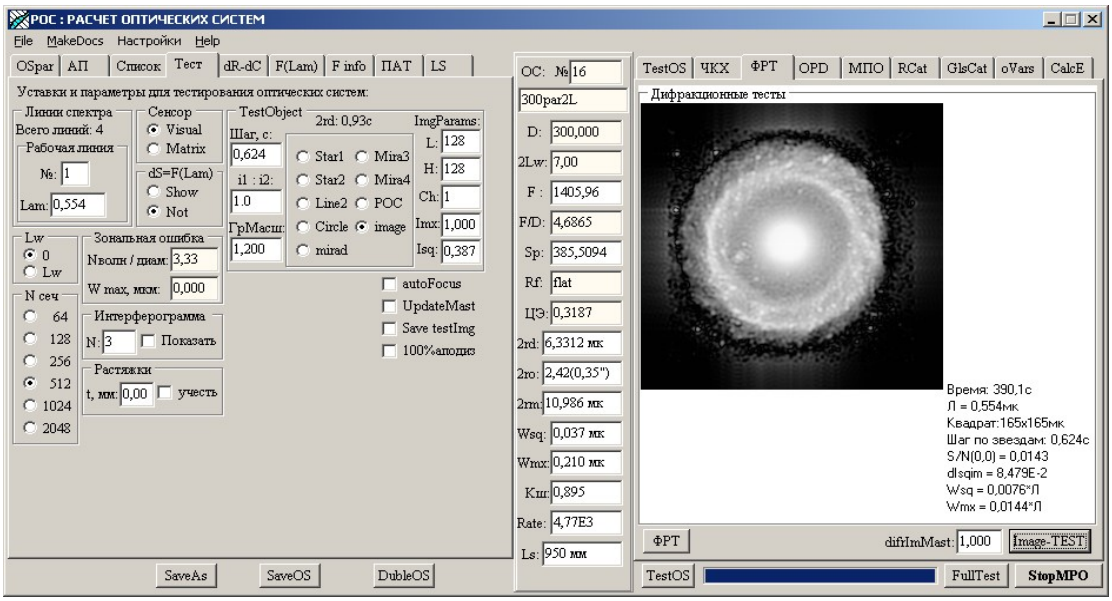


Рис. 6.14. РОС после тестирования реальной системы по тест-объекту.

На втором этапе через реальный объектив формируется дифракционное изображение тест-объекта, которое выводится в окно «Дифракционные тесты» правого блокнота.

Тест-объект должен быть в FIT-файле в формате IEEE Float, а его размеры не превышать 256x256 пикселей, так как программа создает десятки тысяч дублей исходной зрачковой функции тестируемого объектива и при тактовой частоте процессора 1.8 ГГц время обработки одного кадра размером 128x128 пикселей при NSec = 1024 составляет более 10 минут.

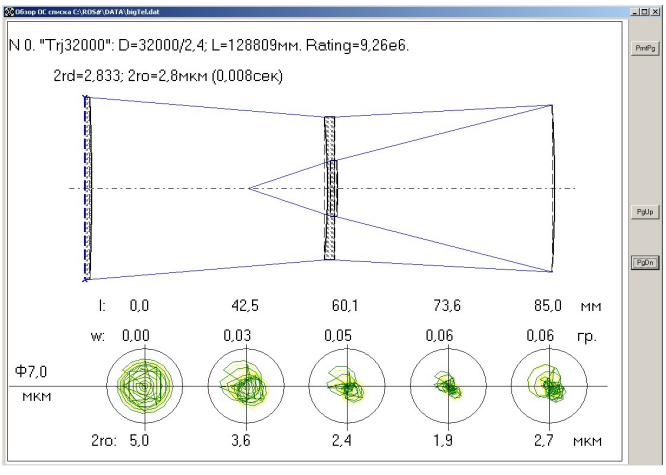
14. МАСШТАБИРОВАНИЕ ОПТИЧЕСКОЙ СИСТЕМЫ

Масштабирование размеров ОС в программе РОС производится посредством изменения в Таблице «внешних» параметров одного из двух внешних параметров: D – диаметра входного зрачка и/или фокделя F/D – фокального делителя. Для вступления в силу этой опции необходимо на странице «Тест» левого блокнота поставить флажок в чекбоксе UpdateMast.

После изменения этих параметров нажмите мышкой на **кнопку «TestOS»**, в результате чего все параметры (кроме размера поля зрения) будут преобразованы пропорционально изменению диаметра ОС или ее фокделя и система будет протестирована.

По окончании процедуры масштабирования, чтобы обеспечить в дальнейшем стабильность измененных параметров, необходимо на странице «Тест» левого блокнота убрать флажок в чекбоксе UpdateMast.

Масштабный эффект



При разработке крупногабаритной оптики обнаруживается линейная зависимость размера пятна рассеяния от габаритных размером ОС. Приходится применять различные ухищрения, чтобы достичь приемлемое качество (см. рис. 11.1). Когда уже ничто не помогает, приходится маскировать

Крупногабаритная система Теребижа.
Хроматизм ограничивает разрешающую способность системы и применение ее возможно только на линиях спектра d и e (желтая и зеленая).

недостатки системы под обтекаемым терминов «субсекундное качество изображения», означающее, что гигантский профессиональный телескоп по качеству изображения уравнивается со школьным.

Масштабирование ОС с использованием МП-оптимизатора

Введите новую ОС или откройте уже существующую и на вкладке MPO произведите следующие изменения:

a. Задайте новый размер максимального пятна, введя его значение в строку 2rm. Эта величина должна быть равна размеру кружка Эйри, который будет давать масштабируемая ОС.
b. Откорректируйте списки оптимизируемых параметров.
c. В боксе «Цель МПО» установите флажок «Удержать» F'.
d. Нажмите кнопку «Старт» MPO.

POC произведет перерасчет радиусов кривизны ОС и их асферичностей, а также осевых расстояний. В результате изменения параметров ОС приобретет фокусное расстояние с равными между собой размерами кружка Эйри и 2rm. После этого программа приступит к оптимизации отмасштабированной таким образом ОС. Тут ее можно притормозить, нажав на кнопку MPO => Stop, и продолжить дальнейшую работу с системой.

15. НОРМАЛИЗАЦИЯ РАЗМЕРОВ РАДИУСОВ

Для изготовления на оптическом заводе радиусы исполнительных поверхностей ОС должны быть нормализованы по каталогу пробных стекол.

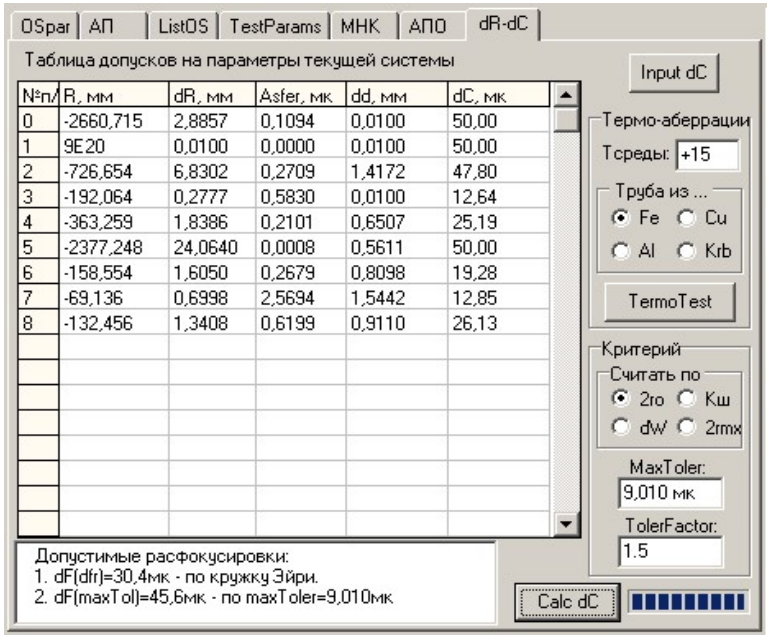
При нажатии **кнопки "NormR"**, расположенной на **вкладке «OSpar»**, POC производит подстановку вместо расчетного значения радиуса – значение ближайшего к нему номинала из **таблицы RCatalog на вкладке «RCatlg»**.

При ухудшении качества изображения вернитесь к первоначальным параметрам, снова выбрав ОС из списка ОС, и, отмасштабировав ее одним из вышеописанных способов, попытайтесь "поймать" качество, наилучшее из возможных.

Если настаиваете на своих номиналах радиусов, то при заказе придется оплачивать изготовление эталонов (пробных стекол), каждый номинал которых обойдется Вам по цене всей Вашей ОС.

Если Вы сами изготавливаете оптику, полагаясь на свои методы контроля, то процедуру нормализации радиусов можно не использовать.

16. РАСЧЕТ ДОПУСКОВ



На **вкладке "dR – dC"** представлена Таблица для отображения значений допусков, под ней – информационное окно и две кнопки для введения «Input dC» и для запуска подпрограммы расчета допусков на параметры ОС «Calc dC».

Вкладка «dR – dC».

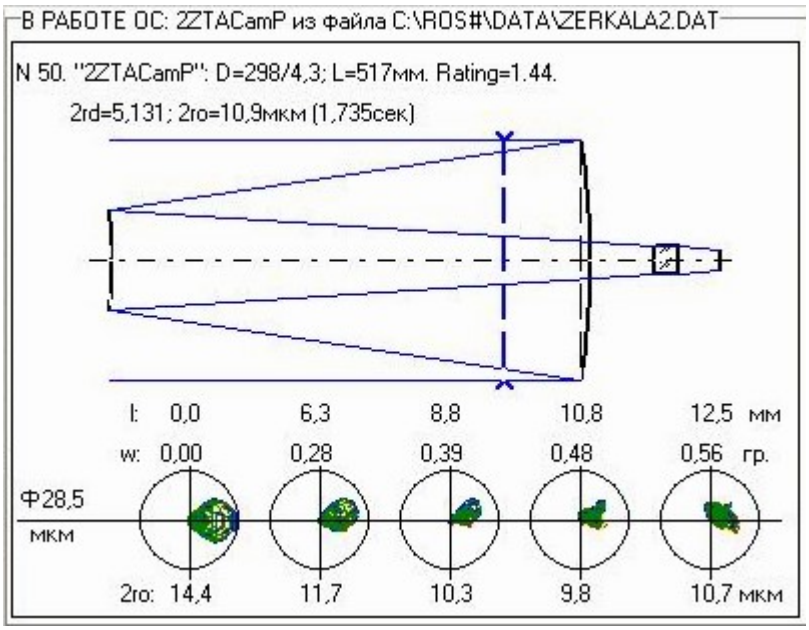
Если пользователю известны ошибки параметров ОС, то он может их внести непосредственно в соответствующие графы таблицы:
dR - ошибка номинала радиуса кривизны поверхности, мм,
Asfer – отступление реальной поверхности (асферичность) от расчетной сферы, мк,
dd - допуск на осевую толщину, мм,
dC - допустимая косина поверхности, мк.

После введения допусков и нажатия на **кнопку "Input dC"** РОС протестирует "косую ОС" и представит результаты тестирования в тест-окне. В **информационном окне вкладки "dR – dC"** появятся значения допустимой расфокусировки для дифракционого и максимального (2rm, выставленного на вкладке МПО) пятен рассеяния.

По нажатии **кнопки "Calc dC"** производится определение расчетных значений допусков.

В качестве критерия принимается одно из значений Максимума – 2ro, 2rmx, Кш или dW, указываемого пользователем в окне редактора «MaxToler».

После введения рассчитанных допусков «косая» ОС тестируется (см. рис. 10.2). При расчете допусков и тестировании производится подфокусировка ОС.



Результат тестирования «косой» ОС.

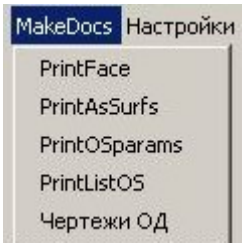
Полученные в результате расчета величины допусков вписываются РОС в таблицы «Требования к изготовлению» при распечатке эскизов оптических деталей.

На этой же вкладке можно оценить влияние изменения температуры на качество изображения (см. рис.10.1, бокс «Термо-аберрации»). После введения ожидаемого перепада температуры и выбора материала трубы телескопа, нажмите кнопку «Termo Test». В процессе термо-тестирования системы программа удерживает неизменным последний отрезок.

17. РАЗРАБОТКА И ПЕЧАТЬ ЭСКИЗОВ ДЕТАЛЕЙ

В программе РОС имеется возможность создания эскизов оптических деталей, входящих в состав ОС, включая склейки.

Войдите в **Меню «MakeDocs»** (см. рис. 9.1) и выберите пункт **"Чертежи ОД"**. Будут сгенерированы стандартные бланки, на которых изображены эскизы деталей и оптическая схема разработанной ОС. Эта документация будет сохранена в формате *.jpg в папке …\ROS#\DATA\DOCS\nameOS (при этом старая папка будет затерта); здесь nameOS – имя ОС, находящейся в данный момент в работе.



Команды Меню «MakeDocs».

Если в Настройках РОС выставлен флажок "Печатать" и Вы не забыли включить принтер, то вся указанная документация будет распечатана на принтере в формате А11 (210x297 мм).

Распечатка на принтере

Оптическая схема и эскизы деталей сохраняются на диске в формате *.jpg и могут быть проанализированы в любом графическом редакторе и распечатаны на принтере.

Однако в программе РОС имеется собственная служба печати.

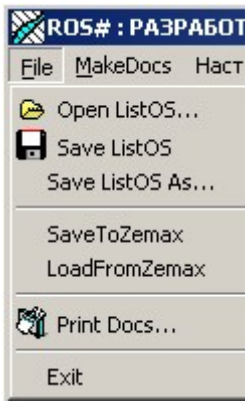
Во-первых, в настройках программы РОС (пункт Меню Настройки) необходимо выставить флажок "Печать".

Во-вторых, в Меню программы MakeDocs (рис. 9.1) выберите одну из четырех команд:

- PrintFace – распечатка Главного окна программы РОС, как есть,
- PrintOSparams – распечатка таблицы внутренних параметров ОС,
- PrintListOS – распечатка списка ОС, входящих в открытый файл,
- Чертежи ОД – сохранение и распечатка эскизов ОД.

18. СВЯЗЬ С ПРОГРАММОЙ ZEMAX

Параметры разработанной ОС могут быть переданы в программу ZeMax. Для этого войдите в Меню File и выберите команду «Save to ZeMax», укажите папку, в которой необходимо сохранить файл с данными ОС и дайте имя этому файлу с расширением nameOS.zmx. Затем, во время работы в программе ZeMax, загрузите сохраненный файл с параметрами своей системы. Поскольку в РОС входной зрачок не является рабочей поверхностью ОС, но, тем не менее, фигурирует, то программой ZeMax он воспринимается, как первая рабочая поверхность ОС. Можно удалить ее, но можно оставить и для зеркальных систем использовать в качестве экрана: параметры экранирования в *.zmx-файл передаются.



Установите параметры функции качества системы по умолчанию, откорректируйте (при необходимости) рабочие длины волн и углы поля зрения и продолжите работу со своей системой в программе ZeMax.

Команды меню для связи РОС с программой ZeMax.

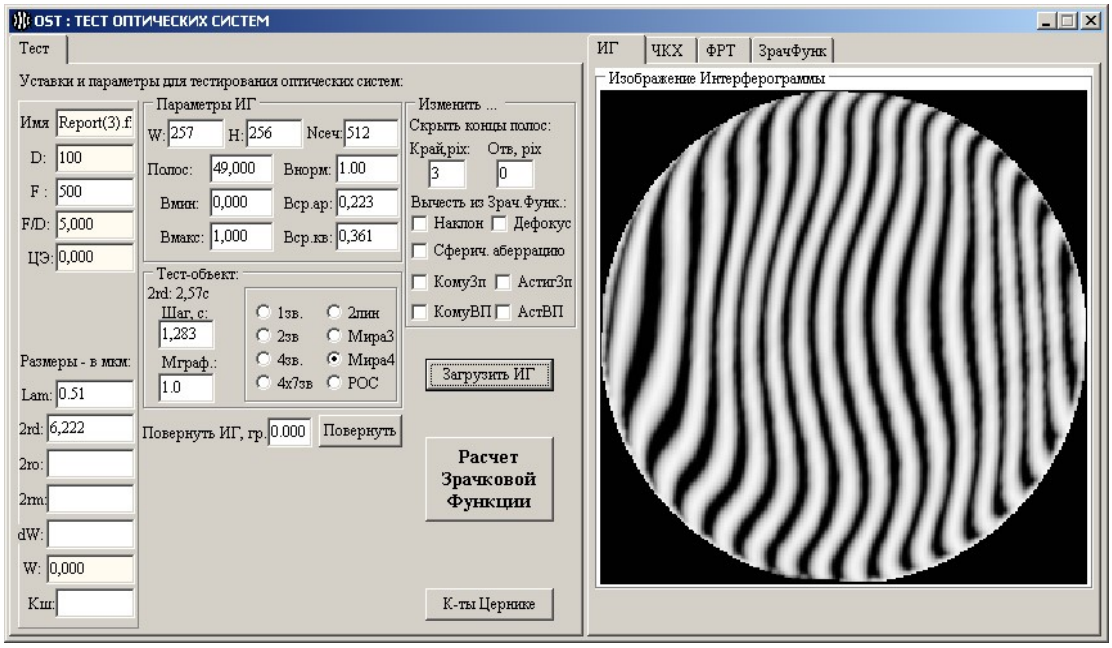
РОС может загрузить ОС, разработанную в программе ZeMax.

Для этого в программе РОС войдите в Меню File и выберите команду «Load from ZeMax», затем выберите папку и *.zmx -файл с искомой ОС.

При этом необходимо иметь в виду, что РОС воспринимает ОС только со стандартными параметрами, но не работает с анаморфотами и прочими ОС. Эксцентриситеты асферических поверхностей в РОС имеют обратные знаки, чем в программе ZeMax. Кроме того, не все каталоги стекол введены в программу РОС, на что и будет указано при загрузке ZeMax -системы.

II. Программа ОСТ

ТЕСТ-ОКНО ПРОГРАММЫ



Внешний вид программы «Оптических систем тестирование»

В левой половине Главного окна расположена вкладка «Тест», на которую выводятся все уставки программы и параметры тестируемой ОС:
- Имя - название оптической системы (по имени открытого файла);
- D – диаметр входного зрачка ОС в миллиметрах;
- F – фокусное расстояние ОС (мм);
- F/D - фокальный делитель;
- ЦЭ - относительная величина центрального экранирования системы;
- dSmx – продольная ошибка ОС, мм (для Фукограмм);
- Lam – длина волны излучения, на которой снята ИГ, (мк);
- 2rd - диаметр дифракционного кружка Эйри для ведущей линии спектра, в микрометрах (мк),
- 2ro – размер осредненного по полю и спектру пятна рассеяния (мк),
- 2rm – размер максимального пятна рассеяния (мк),
- dW – среднеквадратическая величина волновой ошибки ОС (мк);
- W – размах волновой ошибки ОС (мк);
- $K_ш$ – коэффициент Штреля, определенный по ЧКХ.

В этой таблице обязательными к заполнению являются величины D, F, F/D, ЦЭ и Lam. Первые три используются для вычисления вспомогательных параметров ОС, а также для оцифровки координатных осей ЧКХ, ЦЭ – гарантирует корректность считывания данных с ИГ, значение длины волны обеспечивает правильность отображения численных значений волновых ошибок. При качественной оценке ОС ввод параметров D, F, F/D и Lam необязателен.

По нажатии кнопки «Загрузить ИГ» открывается окно выбора файла с изображением ИГ. Программа ОСТ считывает изображение ИГ в форматах *.JPG или *.FIT, но корректная работа пока гарантирована только с форматом *.FIT.

После загрузки в правом блокноте на вкладке «ИГ» появляется изображение ИГ, а на вкладке «Тест» левого блокнота, в боксе «Параметры изображения ИГ» выводятся следующие значения:
- W – ширина изображения ИГ, в пикселях,
- H – высота изображения ИГ, в пикселях,
- Nсеч – число сечений входного зрачка, используемое в процедурах БПФ,
- Вмин – минимальная яркость пикселя на изображении ИГ,
- Вмак – максимальная яркость пикселя на изображении ИГ,
- Вср.ар. – среднеарифметическое значение яркости пикселей на изображении ИГ,

- Вср.кв. – среднеквадратическое значение яркости пикселей на изображении ИГ.
- Внорм – максимальная яркость пикселя на изображении ИГ, использованная для нормирования яркости пикселей по изображению ИГ.

По нажатии кнопки «Расчет Зрачковой Функции» программа ОСТ производит анализ изображения ИГ:
- определяет число ярких полос и в боксе «Параметры изображения ИГ» выводит это значение в окно «Nполос»;
- трассирует полосы, определяя координаты пикселей полосы, и отображает трассу на ИГ. Здесь имеется возможность исправить траекторию трассировок на краю ИГ, для чего в боксе «Изменить …» имеется редактируемое окно – Край, pix – число пикселей на краю ИГ, у которых будет исправлена траектория,
- реконструирует по ИГ зрачковую функцию ОС, используя полиномы Цернике до 14й степени. Результаты этой работы можно посмотреть, нажав на кнопку «К-ты Цернике».

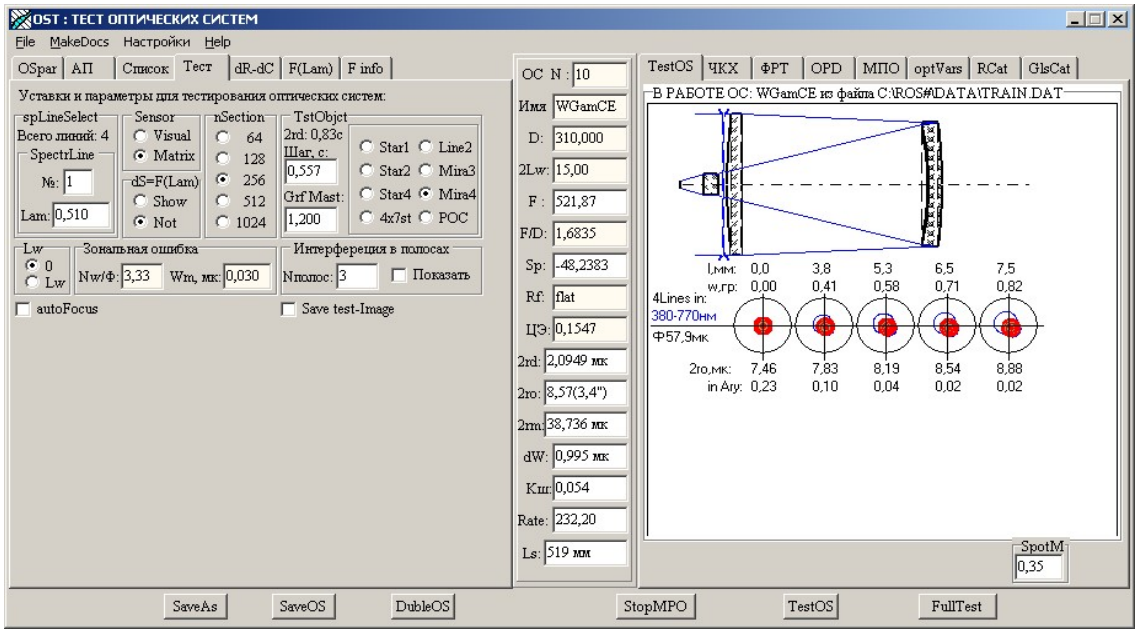
1. Интерферограмма и проблема восстановления фазы

О самом методе аттестации оптики по ИГ1

Точность фиксации распределения освещенности на ИГ ставит предел достоверности оценок качества оптики. Например, ИГ имеет размер 256x256 пикселей и 25 полос (традиционно рекомендуемое число). Тогда на одну полосу приходится 256/25 = 10 пикселей. Следовательно, ошибка в один пиксель даст 1/10 полосы, 1/20 длины волны или, что при Л = 632 нм = 30 нм дает погрешность оценки ошибки системы 0.03 мк.

Давайте оценим, много это или мало, 0.03 мк.

В программе РОС, предназначенной для расчета ОС, на вкладке «Тест» левого блокнота есть опция для оценки влияния зональной ошибки на качество изображения оптической системы:



Бокс «Зональная ошибка» на вкладке «Тест» программы РОС.

Воспользуемся этой возможностью.

Сначала при зональной ошибке Wm = 0 построим ЧКХ системы: Кш = 0.653 (см. рисунок П2). Затем введем Wm = 0.03 мк: Кш = 0.597 (рисунок П3).

При ошибке в 2 пикселя (Л/10 = 0.06 мк) Кш = 0.494 (рисунок П4). Имеет место понижение любимого критерия на четверть (!).

Вот такая печальная арифметика.

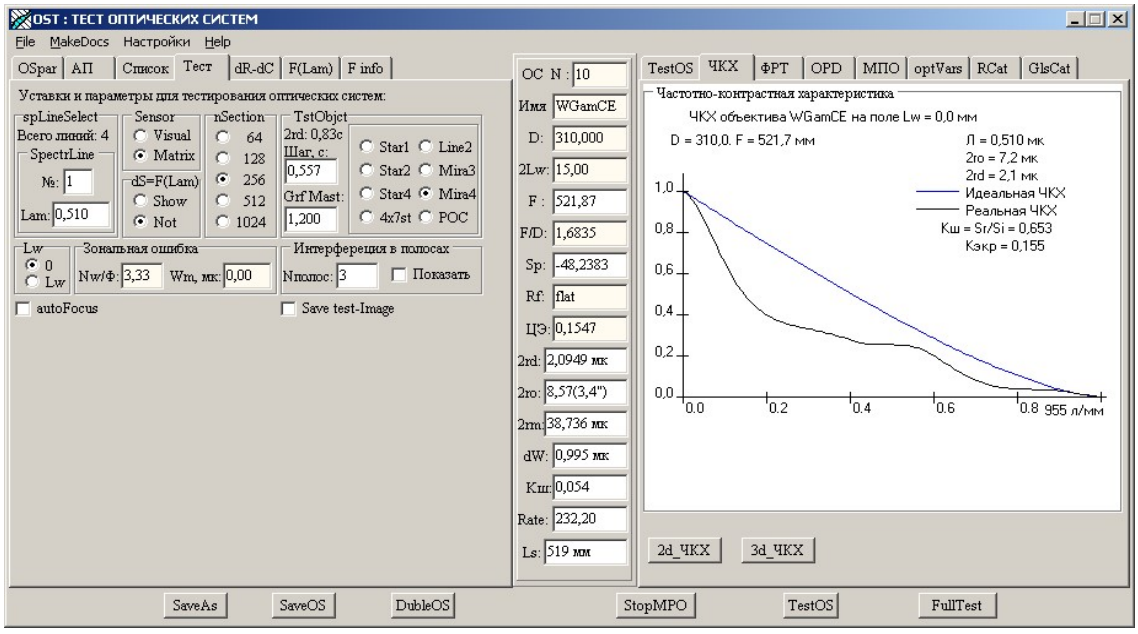


Рис. П2. Исходная ЧКХ и первоначальное значение Кш = 0.653.

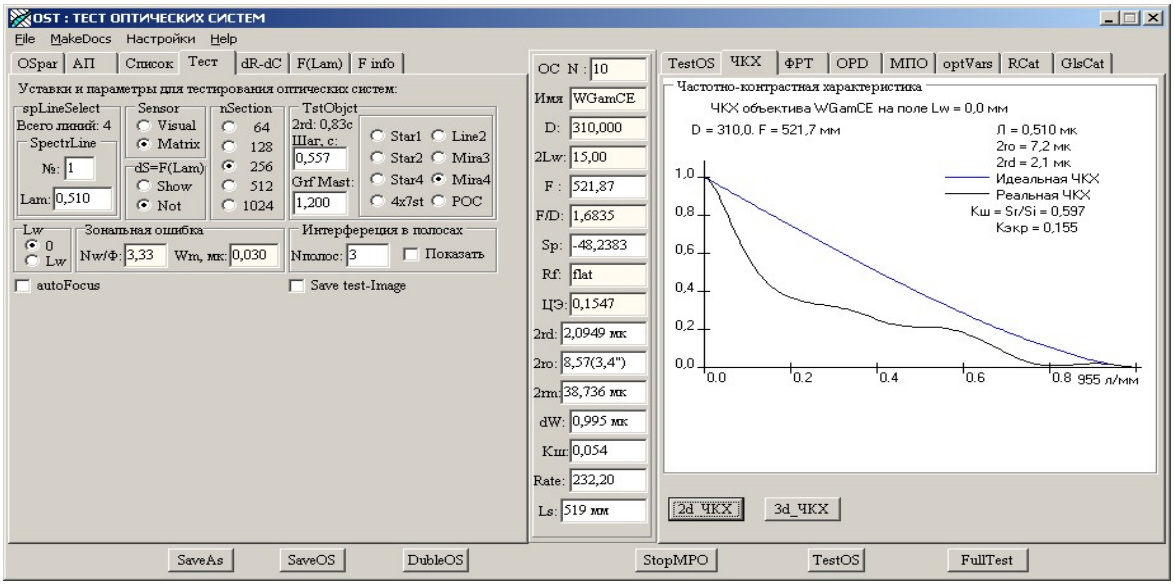


Рис. П3. ЧКХ и Кш при зональной ошибке Wm = 0.03 мк.

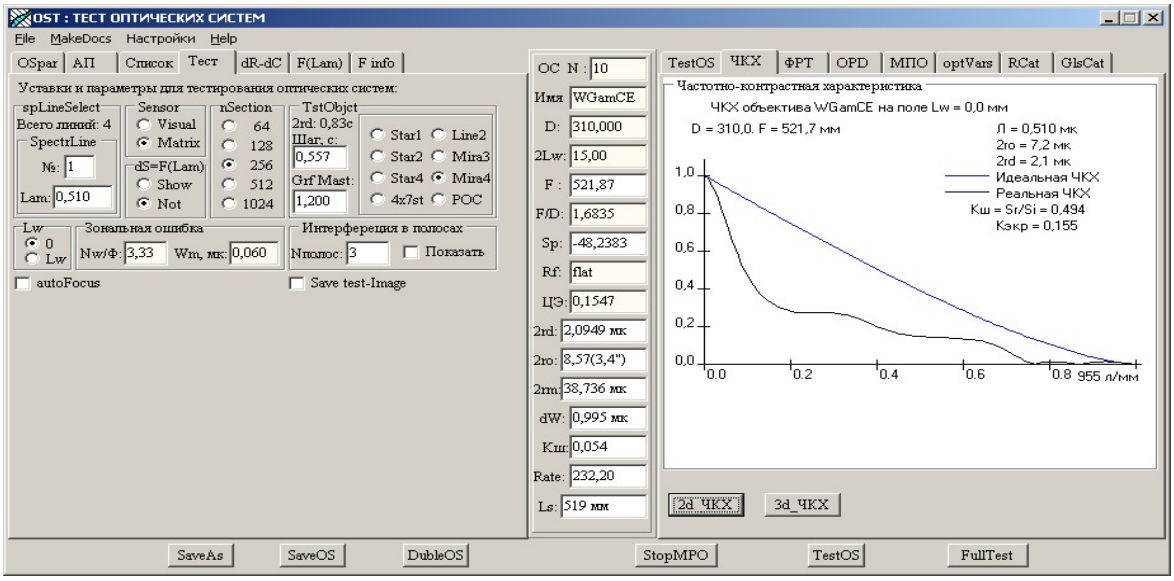


Рис. П4. ЧКХ и Кш при зональной ошибке Wm = 0.06 мк.

Из проведённого анализа следует, что точность оценки ОС можно повысить, уменьшая число полос на ИГ. Однако при этом полосы на краях приобретают значительное искривление, которое при их трассировке вносит большую краевую ошибку. В ОСТ предусмотрена коррекция трассировки, когда до края остается несколько пикселей, и удовлетворительная оценка ОС производится при числе полос от 9 до 20.

Как известно, точность графического решения, например, определения корней уравнений, составляет 5%, т.е. 0.05 = 1/20, что полностью соответствует точности оценки волновой ошибки у ОС в долях длины волны. И, как ни прискорбно, но никакие математические ухищрения (различного типа регрессии, сплайны и т.д. и т.п.) не позволяют улучшить эту ситуацию.

Недавно появилась программа DFTFringe, в которой восстановление фазы волновой ошибки производится не по результатам сканирования полос, а посредством Фурье-преобразования ИГ. Прменение быстрого преобразования Фурье несколько уточняет полученную оценку точности,но вряд ли более, чем вдва раза.

Однако в программе DFTFringe обработке подвергается не один снимок ИГ, а несколько ИГ, отснятых в видео-ролике. Такой подход, в принципе, способен на порядок поднять результирующую точность.

2. Трассировка полос

Первой процедурой при обработке изображения интерференционной картины, называемой интерферограммой или сокращённо ИГ, является определение числа вертикальных полос в пределах измеряемого диаметра и формирование массива с координатами центров полос в срединном сечении ВЗ.

Найденные координаты центров полос используются на следующем этапе для попиксельного движения вдоль соотвествующих полос.

После нахождения очередного максимума на полосе его координаты из целочисленного значения, путём использования значения яркостей соседних пикселей, переводятся в действительное значение посредством применения синусоидальной аппроксимации графика яркостей в поперечном сечении полосы.

Если у аттестуемой ОС нет центрального экранирования (ЦЭ), то на ИГ отсутствуют разрывы интерференционных полос. В этом случае используется следующая функция.

```
function DefPolos:float;
// число вертикальных полос в горизонтальном сечении в ОС без ЦЭ
var
 i,j,k,ibn,ied,pmx,pmn: integer;
 a1,a2,x1,x2: float;
begin
 i:=0;
 k:=q;
 pmx:=0;
 pmn:=0;
 j:=k*w;
 a1:=Crooks^[j+i];
 a2:=Crooks^[j+i+1];
 inc(i);
 while i<=w do
 if a2>a1 then
  repeat
   a1:=a2;
   a2:=Crooks^[j+i];
   if a2<a1 then
    begin
     mxPosV[pmx].x:=i;
     mxPosV[pmx].y:=q;
     inc(pmx);
    end;
   inc(i);
  until ((a2<a1) or (i=w))
 else
  repeat
   a1:=a2;
   a2:=Crooks^[j+i];
   if a2>a1 then
    begin
     mnPos[pmn].x:=i;
     mnPos[pmx].x:=q;
     inc(pmn);
    end;
   inc(i);
```

```
  until ((a2>a1) or (i=w));
  i:=mnPos[0].x;
  j:=mxPosV[0].x;
  if i<j then
   ibn:=i
  else
   ibn:=j;
  i:=mnPos[pmn].x;
  if i>w then
   i:=w;
  j:=mxPosV[pmx].x;
  if j>w then
   j:=w;
  if i>j then
   ied:=i
  else
   ied:=j;
  result:=pmx+1-(ied-ibn)/w;
end;
```

Если же ОС имеет ЦЭ, о чём пользователь программы ОСТ должен ей сообщить, введя в редакционное окно «ЦЭ» число Cekr, соответствующее величине ЦЭ в относительных единицах от диаметра ВЗ контролируемой ОС.

В этом случае программа ОСТ использует более сложный алгоритм подсчета числа полос.

На первом этапе производится сканирование посередине ИГ поперёк полос и подсчёт их числа, начиная с левого края ВЗ, до левого края центрального экрана ОС.

На втором этапе производится сканирование и подсчёт числа полос на уровне верхнего края ЦЭ, начиная от левого края центрального экрана и заканчивая правым краем ЦЭ.

На третьем этапе сканируются и подсчитываются полосы на уровне нижнего края ЦЭ, также в пределах центрального экрана.

На последнем, четвёртом, этапе программа заканчивает подсчёт полного числа полос, сканируя поперёк полос посередине ИГ от правого края ЦЭ до правого края ВЗ.

Процедуры DefVKrayCE и DefNKrayCE используются для подсчёта числа полос, соответственно, на уровне верхнего и нижнего краёв ЦЭ.

Чтобы при поперечном сканировании полос избежать случайного выхода в заэкранированную зону, сканирование полос производится на 2 пикселя выше/ниже ЦЭ. Именно поэтому верхняя граница ЦЭ ограничена величиной 0.7, т.е Cekr должно быть не более 0.7.

```
procedure DefVKrayCE(
 xv: integer;   // xBeg of Crook
 var x: integer // xEnd of Crook
);
var
 i,k,qe: integer;
 a1,a2,a3: float;
begin
 k:=xv;
 i:=q;
 qe:=q-qce-1;
 while (i>qe) do
 begin
  a1:=crooks^[w*i+k-1];
  a2:=crooks^[w*i+k];
```

```
  a3:=crooks^[w*i+k+1];
  if ((a1>a2) and (a1>a3)) then
   dec(k);
  if ((a3>a1) and (a3>a2)) then
   inc(k);
   dec(i);  // на пиксель выше
 end;
 x:=k;
end;

procedure DefNKrayCE(
 xv: integer;   // xBeg of Crook
 var x: integer // xEnd of Crook
);
var
 i,k,qe: integer;
 a1,a2,a3: float;
begin
 k:=xv;
 i:=q;
 qe:=q+qce+1;
 while (i<qe) do
 begin
  a1:=crooks^[w*i+k-1];
  a2:=crooks^[w*i+k];
  a3:=crooks^[w*i+k+1];
  if ((a1>a2) and (a1>a3)) then
   dec(k);
  if ((a3>a1) and (a3>a2)) then
   inc(k);
  inc(i); // на пиксель ниже
 end;
 x:=k;
end;

function DefFullPlsV:float;
// число вертикальных полос над ЦЭ: Goriz/Circ/Goriz
var
 i,j,k,m,n,x2,x3,y,
  np1,np2,np3,ib,ied,ibn,ibx,ien,iex,pmx,pmn,qe: integer;
 a1,a2: float;
 mxPos2,mxPos3: array [0..49] of IPoint;
begin
// от левого края ВЗ до правого края ЦЭ
 i:=0;
 pmx:=0;
 pmn:=0;
 j:=q*w;
 qe:=q-qce;   // левый край ЦЭ
 a1:=Crooks^[j];
 a2:=Crooks^[j+1];
 inc(i);
 while (i<=qe) do
 if a2>a1 then
```

```
  repeat
   a1:=a2;
   a2:=Crooks^[j+i];
   if a2<a1 then
    begin
     mxPosV[pmx].x:=i;
     mxPosV[pmx].y:=q;
     inc(pmx);
    end;
   inc(i);
  until ((a2<a1) or (i>qe))
 else
  repeat
   a1:=a2;
   a2:=Crooks^[j+i];
   if a2>a1 then
    begin
     mnPos[pmn].x:=i;
     mnPos[pmn].y:=q;
     inc(pmn);
    end;
   inc(i);
  until ((a2>a1) or (i>qe));
 np1:=pmx;
 ibn:=mnPos[0].x;
 ibx:=mxPosV[0].x;
// левый узел над ЦЭ:
 if qce>0 then
  DefVKrayCE(mxPosV[np1-1].x,x2);
// от правого края ЦЭ до правого края ВЗ
 pmx:=0;
 pmn:=0;
 i:=q+qce;
 j:=q*w;
 a1:=Crooks^[j+i];
 a2:=Crooks^[j+i+1];
 inc(i);
 while (i<w) do
 if a2>a1 then
  repeat
   a1:=a2;
   a2:=Crooks^[j+i];
   if a2<a1 then
    begin
     mxPos3[pmx].x:=i;
     mxPos3[pmx].y:=q;
     inc(pmx);
    end;
   inc(i);
  until ((a2<a1) or (i=w))
 else
  repeat
   a1:=a2;
   a2:=Crooks^[j+i];
```

```
   if a2>a1 then
    begin
     mnPos[pmn].x:=i;
     mnPos[pmn].y:=q;
     inc(pmn);
    end;
   inc(i);
  until ((a2>a1) or (i=w));
 np3:=pmx;
 ien:=mnPos[pmn-1].x;
 iex:=mxPos3[pmx-1].x;
 if qce>0 then
 begin
 // правый узел над ЦЭ:
  DefVKrayCE(mxPos3[0].x,x3);
// от левого до правого края и над ЦЭ, параллельно оси
  pmx:=0;
  pmn:=0;
  qe:=x3-1;
  y:=q-qce-1;
  i:=x2+1;
  j:=y*w;
  a1:=Crooks^[j+i];
  a2:=Crooks^[j+i+1];
  inc(i);
  while (i<qe) do
  if a2>a1 then
   repeat
    a1:=a2;
    a2:=Crooks^[j+i];
    if a2<a1 then
    begin
     mxPos2[pmx].x:=i;
     mxPos2[pmx].y:=y;
     inc(pmx);
    end;
    inc(i);
   until ((a2<a1) or (i=qe))
  else
   repeat
    a1:=a2;
    a2:=Crooks^[j+i];
    if a2>a1 then
    begin
     mnPos[pmn].x:=i;
     mnPos[pmn].y:=y;
     inc(pmn);
    end;
    inc(i);
   until ((a2>a1) or (i=qe));
  np2:=pmx;
  for i:=0 to np2-1 do
  begin
   mxPosV[np1+i].x:=mxPos2[i].x;
```

```
   mxPosV[np1+i].y:=mxPos2[i].y;
  end;
 end
 else
  np2:=0;
 for i:=0 to np3-1 do
 begin
  mxPosV[np1+np2+i].x:=mxPos3[i].x;
  mxPosV[np1+np2+i].y:=mxPos3[i].y;
 end;
  pmx:=np1+np2+np3;
  i:=ibn;
  j:=ibx;
  if i<j then
   ibn:=i
  else
   ibn:=j;
  i:=ien;
  if i>w then
   i:=w;
  j:=iex;
  if j>w then
   j:=w;
  if i>j then
   ied:=i
  else
   ied:=j;
  result:=pmx+1-(ied-ibn)/w;
end;

function DefFullPlsN:float;
// число вертикальных полос под ЦЭ: Goriz/Circ/Goriz
var
 i,j,k,m,n,x2,x3,y,
  np1,np2,np3,ib,ied,ibn,ibx,ien,iex,pmx,pmn,qe: integer;
 a1,a2: float;
begin
// от левого края ВЗ до правого края ЦЭ
 i:=0;
 pmx:=0;
 pmn:=0;
 j:=q*w;
 qe:=q-qce;   // левый край ЦЭ
 a1:=Crooks^[j];
 a2:=Crooks^[j+1];
 inc(i);
 while (i<=qe) do
 if a2>a1 then
  repeat
   a1:=a2;
   a2:=Crooks^[j+i];
   if a2<a1 then
    begin
     mxPosN[pmx].x:=i;
```

```
      mxPosN[pmx].y:=q;
      inc(pmx);
     end;
    inc(i);
   until ((a2<a1) or (i>qe))
  else
   repeat
    a1:=a2;
    a2:=Crooks^[j+i];
    if a2>a1 then
     begin
      mnPos[pmn].x:=i;
      mnPos[pmn].y:=q;
      inc(pmn);
     end;
    inc(i);
   until ((a2>a1) or (i>qe));
  np1:=pmx;
  ibn:=mnPos[0].x;
  ibx:=mxPosN[0].x;
// левый узел под ЦЭ:
  if qce>0 then
   DefNKrayCE(mxPosN[np1-1].x,x2);
// от правого края ЦЭ до правого края ВЗ
  pmx:=0;
  pmn:=0;
  i:=q+qce;
  j:=q*w;
  a1:=Crooks^[j+i];
  a2:=Crooks^[j+i+1];
  inc(i);
  while (i<w) do
  if a2>a1 then
   repeat
    a1:=a2;
    a2:=Crooks^[j+i];
    if a2<a1 then
     begin
      mxPos3[pmx].x:=i;
      mxPos3[pmx].y:=q;
      inc(pmx);
     end;
    inc(i);
   until ((a2<a1) or (i=w))
  else
   repeat
    a1:=a2;
    a2:=Crooks^[j+i];
    if a2>a1 then
     begin
      mnPos[pmn].x:=i;
      mnPos[pmn].y:=q;
      inc(pmn);
     end;
```

```
   inc(i);
  until ((a2>a1) or (i=w));
 np3:=pmx;
 ien:=mnPos[pmn-1].x;
 iex:=mxPos3[pmx-1].x;
// правый узел под ЦЭ:
 if qce>0 then
 begin
  DefNKrayCE(mxPos3[0].x,x3);
// от левого до правого края и под ЦЭ, параллельно оси
  pmx:=0;
  pmn:=0;
  qe:=x3-1;
  y:=q+qce+1;
  i:=x2+1;
  j:=y*w;     // под ЦЭ
  a1:=Crooks^[j+i];
  a2:=Crooks^[j+i+1];
  inc(i);
  while (i<qe) do
  if a2>a1 then
   repeat
    a1:=a2;
    a2:=Crooks^[j+i];
    if a2<a1 then
    begin
     mxPos2[pmx].x:=i;
     mxPos2[pmx].y:=y;
     inc(pmx);
    end;
    inc(i);
   until ((a2<a1) or (i=qe))
  else
   repeat
    a1:=a2;
    a2:=Crooks^[j+i];
    if a2>a1 then
    begin
     mnPos[pmn].x:=i;
     mnPos[pmn].y:=y;
     inc(pmn);
    end;
    inc(i);
   until ((a2>a1) or (i=qe));
   np2:=pmx;
   for i:=0 to np2-1 do
   begin
    mxPosN[np1+i].x:=mxPos2[i].x;
    mxPosN[np1+i].y:=mxPos2[i].y;
   end;
  end
 else
  np2:=0;
 for i:=0 to np3-1 do
```

```
 begin
  mxPosN[np1+np2+i].x:=mxPos3[i].x;
  mxPosN[np1+np2+i].y:=mxPos3[i].y;
 end;
 pmx:=np1+np2+np3;
 i:=ibn;
 j:=ibx;
 if i<j then
  ibn:=i
 else
  ibn:=j;
 i:=ien;
 if i>w then
  i:=w;
 j:=iex;
 if j>w then
  j:=w;
 if i>j then
  ied:=i
 else
  ied:=j;
 result:=pmx+1-(ied-ibn)/w;
end;
```

После получения для полос координат «вершин хребтов» производится сканирование вдоль полос, начиная – в отсутствие ЦЭ – со средины полосы, в сторону её верхнего и нижнего концов. В случае ЦЭ сканирование вдоль полосы производится, начиная от конца, прилегающего к кромке ЦЭ, в сторону края ВЗ.

3. ПРИМЕНЕНИЕ ПОЛИНОМОВ ЦЕРНИКЕ И МЕТОДА НАИМЕНЬШИХ КВАДРАТОВ ПРИ ВОССТАНОВЛЕНИИ ЗРАЧКОВОЙ ФУНКЦИИ ОС

В результате сканироания N полос образуется N массивов, содержащих координаты «вершин хребтов». Задачей дальнейших преобразованй является восстановление по искривлению интерференционных полос формы зрачковой функции ОС, т.е. величины волновых ошибок и их распределения по ВЗ.

Для этого целесообразно использовать некую априорную универсальную функцию, какую, например, мы использовали при описании асферических поверхностей (обобщенный профиль Кербера). Однако профиль Кербера основан на степенном базисе ρ^{2k}, где $\rho = y/H$ – относительная ордината на оптической поверхности, а 2k – представляет собой её четную степень. С помощью степенного базиса удобно описывать осесимметричные ошибки, но затруднительно – более сложные отклонения волнового фронта.

Целесообразнее оказываются полиномы Цернике. Они составляют ортогональный базис, особенностью которого является независимость друг от друга полиномов, описывающих разнородные искажения волнового фронта – комы, астигматизма и др. Это позволяет после восстановления зрачковой функции ОС производить её поаберрационный анализ путём выборочного вычленения той или иной аберрации.

Определение коэффициентов, входящих в состав полиномов Цернике, в программе ОСТ производится методом наименьших квадратов (МНК).

На первом этапе формируется информационная матрица, состоящая из членов, полученных преобразованием искривления интерференционных полос в первоначальное, приближённое, значение волновой ошибки. Назначение МНК состоит в том, чтобы путём статистической обработки и осреднения экспериментальных данных, рафинировать результат, значительно повысив его точность.

Затем вычисляется детерминант информационной матрицы и значения коэффициентов Цернике. В конце производится коррекция коэффициентов с учётом их весов.

Ниже приведён комплект функций kc11 – ks122 и ks11 – ks122, вычисляющих значения соответствующих полиномов Цернике, как при формировании информационной матрицы для МНК, так и при вычислении непосредственных значений зрачковой функции исследуемой ОС.

```
function kc11(r,cs:float):float;
begin
 result:=r*cs;
end;

function ks11(r,sn:float):float;
begin
 result:=r*sn;
end;

function kc20(r2:float):float;
begin
 result:=(r2*2-1);
end;

function kc22(r2,cs2:float):float;
begin
 result:=r2*cs2;
end;
```

```
function ks22(r2,sn2:float):float;
begin
 result:=r2*sn2;
end;

function kc31(r,r2,cs:float):float;
begin
 result:=(r2*3-2)*r*cs;
end;

function ks31(r,r2,sn:float):float;
begin
 result:=(r2*3-2)*r*sn;
end;

function kc33(r,r2,cs3:float):float;
begin
 result:=r*r2*cs3;
end;

function ks33(r,r2,sn3:float):float;
begin
 result:=r*r2*sn3;
end;

function kc40(r2:float):float;
begin
 result:=((r2*6-6)*r2+1);
end;

function kc42(r2,cs2:float):float;
begin
 result:=(r2*4-3)*r2*cs2;
end;

function ks42(r2,sn2:float):float;
begin
 result:=(r2*4-3)*r2*sn2;
end;

function kc44(r2,cs4:float):float;
begin
 result:=sqr(r2)*cs4;
end;

function ks44(r2,sn4:float):float;
begin
 result:=sqr(r2)*sn4;
end;

function kc51(r,r2,cs:float):float;
begin
 result:=((r2*10-12)*r2+3)*r*cs;
```

```
end;

function ks51(r,r2,sn:float):float;
begin
 result:=((r2*10-12)*r2+3)*r*sn;
end;

function kc53(r,r2,cs3:float):float;
begin
 result:=r*r2*(r2*5-4)*cs3;
end;

function ks53(r,r2,sn3:float):float;
begin
 result:=r*r2*(r2*5-4)*sn3;
end;

function kc55(r,r2,cs5:float):float;
begin
 result:=sqr(r2)*r*cs5;
end;

function ks55(r,r2,sn5:float):float;
begin
 result:=sqr(r2)*r*sn5;
end;

function kc60(r2:float):float;
begin
 result:=(((r2*20-30)*r2+12)*r2-1);
end;

function kc62(r2,cs2:float):float;
begin
 result:=((r2*15-20)*r2+6)*r2*cs2;
end;

function ks62(r2,sn2:float):float;
begin
 result:=((r2*15-20)*r2+6)*r2*sn2;
end;

function kc64(r2,cs4:float):float;
begin
 result:=(r2*6-5)*sqr(r2)*cs4;
end;

function ks64(r2,sn4:float):float;
begin
 result:=(r2*6-5)*sqr(r2)*sn4;
end;

function kc66(r2,cs6:float):float;
begin
```

```
 result:=r2*sqr(r2)*cs6;
end;

function ks66(r2,sn6:float):float;
begin
 result:=r2*sqr(r2)*sn6;
end;

function kc71(r,r2,cs:float):float;
begin
 result:=(((r2*35-60)*r2+30)*r2-4)*r*cs;
end;

function ks71(r,r2,sn:float):float;
begin
 result:=(((r2*35-60)*r2+30)*r2-4)*r*sn;
end;

function kc73(r,r2,cs3:float):float;
begin
 result:=((r2*21-30)*r2+10)*r2*r*cs3;
end;

function ks73(r,r2,sn3:float):float;
begin
 result:=((r2*21-30)*r2+10)*r2*r*sn3;
end;

function kc75(r,r2,cs5:float):float;
begin
 result:=(r2*7-6)*sqr(r2)*r*cs5;
end;

function ks75(r,r2,sn5:float):float;
begin
 result:=(r2*7-6)*sqr(r2)*r*sn5;
end;

function kc77(r,r2,cs7:float):float;
begin
 result:=r2*sqr(r2)*r*cs7;
end;

function ks77(r,r2,sn7:float):float;
begin
 result:=r2*sqr(r2)*r*sn7;
end;

function kc80(r2:float):float;
begin
 result:=(((r2*70-140)*r2+90)*r2-20)*r2+1);
end;

function kc82(r2,cs2:float):float;
```

```
begin
 result:=(((r2*56-105)*r2+60)*r2-10)*r2*cs2;
end;

function ks82(r2,sn2:float):float;
begin
 result:=(((r2*56-105)*r2+60)*r2-10)*r2*sn2;
end;

function kc84(r2,cs4:float):float;
begin
 result:=((r2*28-42)*r2+15)*sqr(r2)*cs4;
end;

function ks84(r2,sn4:float):float;
begin
 result:=((r2*28-42)*r2+15)*sqr(r2)*sn4;
end;

function kc86(r2,cs6:float):float;
begin
 result:=(r2*8-7)*sqr(r2)*r2*cs6;
end;

function ks86(r2,sn6:float):float;
begin
 result:=(r2*8-7)*sqr(r2)*r2*sn6;
end;

function kc91(r,r2,cs:float):float;
begin
 result:=((((r2*126-280)*r2+210)*r2-60)*r2+5)*r*cs;
end;

function ks91(r,r2,sn:float):float;
begin
 result:=((((r2*126-280)*r2+210)*r2-60)*r2+5)*r*sn;
end;

function kc93(r,r2,cs3:float):float;
begin
 result:=(((r2*84-168)*r2+105)*r2-20)*r2*r*cs3;
end;

function ks93(r,r2,sn3:float):float;
begin
 result:=(((r2*84-168)*r2+105)*r2-20)*r2*r*sn3;
end;

function kc95(r,r2,cs5:float):float;
begin
 result:=((r2*36-56)*r2+21)*sqr(r2)*r*cs5;
end;
```

```
function ks95(r,r2,sn5:float):float;
begin
 result:=((r2*36-56)*r2+21)*sqr(r2)*r*sn5;
end;

function kc100(r2:float):float;
begin
 result:=((((r2*252-630)*r2+560)*r2-210)*r2+30)*r2-1);
end;

function kc120(r2:float):float;
begin
 result:=(((((r2*924-2772)*r2+3150)*r2-1680)*r2+420)*r2-42)*r2+1);
end;

function kc140(r2:float):float;
begin
 result:=((((((r2*3432-12012)*r2+16632)*r2-11550)*r2+4200)*r2-
756)*r2+56)*r2-1);
end;

function kc111(r,r2,cs:float):float;
begin
 result:=((((r2*462-1260)*r2+1260)*r2-560)*r2+105)*r2-6)*r*cs;
end;

function ks111(r,r2,sn:float):float;
begin
 result:=((((r2*462-1260)*r2+1260)*r2-560)*r2+105)*r2-6)*r*sn;
end;

function kc113(r,r2,cs3:float):float;
begin
 result:=(((r2*330-840)*r2+756)*r2-280)*r2+35)*r2*r*cs3;
end;

function ks113(r,r2,sn3:float):float;
begin
 result:=(((r2*330-840)*r2+756)*r2-280)*r2+35)*r2*r*sn3;
end;

function kc131(r,r2,cs:float):float;
begin
 result:=(((((r2*1716-5544)*r2+6930)*r2-4200)*r2+1260)*r2-
168)*r2+7)*r*cs;
end;

function ks131(r,r2,sn:float):float;
begin
 result:=(((((r2*1716-5544)*r2+6930)*r2-4200)*r2+1260)*r2-
168)*r2+7)*r*sn;
end;

function kc102(r2,cs2:float):float;
```

```
begin
 result:=((((r2*210-504)*r2+420)*r2-140)*r2+15)*r2*cs2;
end;

function ks102(r2,sn2:float):float;
begin
 result:=((((r2*210-504)*r2+420)*r2-140)*r2+15)*r2*sn2;
end;

function kc104(r2,cs4:float):float;
begin
 result:=(((r2*120-252)*r2+168)*r2-35)*sqr(r2)*cs4;
end;

function ks104(r2,sn4:float):float;
begin
 result:=(((r2*120-252)*r2+168)*r2-35)*sqr(r2)*sn4;
end;

function kc122(r2,cs2:float):float;
begin
 result:=((((r2*792-2310)*r2+2520)*r2-1260)*r2+280)*r2-21)*r2*cs2;
end;

function ks122(r2,sn2:float):float;
begin
 result:=((((r2*792-2310)*r2+2520)*r2-1260)*r2+280)*r2-21)*r2*sn2;
end;

// вычисление коэффициентов Цернике методом МНК
procedure CSZernike;
var
 wk, // dW на точке в долях полосы
 ro,ro2, // r/H0, r2/H2
 cs,cs2,cs3,cs4,cs5,cs6,cs7, // sin/cos(N*fi)
 sn,sn2,sn3,sn4,sn5,sn6,sn7: float;
 sv: PVector; // colomn of free
 pH: PMatrix; // matrix

procedure Moderow(k:integer;pr:float);
begin
 pH^[k]^[0]:=pH^[k]^[0]+pr;
 pH^[k]^[1]:=pH^[k]^[1]+kc11(ro,cs)*pr;
 pH^[k]^[2]:=pH^[k]^[2]+ks11(ro,sn)*pr;
 pH^[k]^[3]:=pH^[k]^[3]+kc20(ro2)*pr;
 pH^[k]^[4]:=pH^[k]^[4]+kc22(ro2,cs2)*pr;
 pH^[k]^[5]:=pH^[k]^[5]+ks22(ro2,sn2)*pr;
 pH^[k]^[6]:=pH^[k]^[6]+kc31(ro,ro2,cs)*pr;
 pH^[k]^[7]:=pH^[k]^[7]+ks31(ro,ro2,sn)*pr;
 pH^[k]^[8]:=pH^[k]^[8]+kc33(ro,ro2,cs3)*pr;
 pH^[k]^[9]:=pH^[k]^[9]+ks33(ro,ro2,sn3)*pr;
 pH^[k]^[10]:=pH^[k]^[10]+kc40(ro2)*pr;
 pH^[k]^[11]:=pH^[k]^[11]+kc42(ro2,cs2)*pr;
 pH^[k]^[12]:=pH^[k]^[12]+ks42(ro2,sn2)*pr;
```

```
 pH^[k]^[13]:=pH^[k]^[13]+kc44(ro2,cs4)*pr;
 pH^[k]^[14]:=pH^[k]^[14]+ks44(ro2,sn4)*pr;
 pH^[k]^[15]:=pH^[k]^[15]+kc51(ro,ro2,cs)*pr;
 pH^[k]^[16]:=pH^[k]^[16]+ks51(ro,ro2,sn)*pr;
 pH^[k]^[17]:=pH^[k]^[17]+kc53(ro,ro2,cs3)*pr;
 pH^[k]^[18]:=pH^[k]^[18]+ks53(ro,ro2,sn3)*pr;
 pH^[k]^[19]:=pH^[k]^[19]+kc55(ro,ro2,cs5)*pr;
 pH^[k]^[20]:=pH^[k]^[20]+ks55(ro,ro2,sn5)*pr;
 pH^[k]^[21]:=pH^[k]^[21]+kc60(ro2)*pr;
 pH^[k]^[22]:=pH^[k]^[22]+kc62(ro2,cs2)*pr;
 pH^[k]^[23]:=pH^[k]^[23]+ks62(ro2,sn2)*pr;
 pH^[k]^[24]:=pH^[k]^[24]+kc64(ro2,cs4)*pr;
 pH^[k]^[25]:=pH^[k]^[25]+ks64(ro2,sn4)*pr;
 pH^[k]^[26]:=pH^[k]^[26]+kc66(ro2,cs6)*pr;
 pH^[k]^[27]:=pH^[k]^[27]+ks66(ro2,sn6)*pr;
 pH^[k]^[28]:=pH^[k]^[28]+kc71(ro,ro2,cs)*pr;
 pH^[k]^[29]:=pH^[k]^[29]+ks71(ro,ro2,sn)*pr;
 pH^[k]^[30]:=pH^[k]^[30]+kc73(ro,ro2,cs3)*pr;
 pH^[k]^[31]:=pH^[k]^[31]+ks73(ro,ro2,sn3)*pr;
 pH^[k]^[32]:=pH^[k]^[32]+kc75(ro,ro2,cs5)*pr;
 pH^[k]^[33]:=pH^[k]^[33]+ks75(ro,ro2,sn5)*pr;
 pH^[k]^[34]:=pH^[k]^[34]+kc77(ro,ro2,cs7)*pr;
 pH^[k]^[35]:=pH^[k]^[35]+ks77(ro,ro2,sn7)*pr;
 pH^[k]^[36]:=pH^[k]^[36]+kc80(ro2)*pr;
 pH^[k]^[37]:=pH^[k]^[37]+kc82(ro2,cs2)*pr;
 pH^[k]^[38]:=pH^[k]^[38]+ks82(ro2,sn2)*pr;
 pH^[k]^[39]:=pH^[k]^[39]+kc84(ro2,cs4)*pr;
 pH^[k]^[40]:=pH^[k]^[40]+ks84(ro2,sn4)*pr;
 pH^[k]^[41]:=pH^[k]^[41]+kc86(ro2,cs6)*pr;
 pH^[k]^[42]:=pH^[k]^[42]+ks86(ro2,sn6)*pr;
 pH^[k]^[43]:=pH^[k]^[43]+kc91(ro,ro2,cs)*pr;
 pH^[k]^[44]:=pH^[k]^[44]+ks91(ro,ro2,sn)*pr;
 pH^[k]^[45]:=pH^[k]^[45]+kc93(ro,ro2,cs3)*pr;
 pH^[k]^[46]:=pH^[k]^[46]+ks93(ro,ro2,sn3)*pr;
 pH^[k]^[47]:=pH^[k]^[47]+kc95(ro,ro2,cs5)*pr;
 pH^[k]^[48]:=pH^[k]^[48]+ks95(ro,ro2,sn5)*pr;
 pH^[k]^[49]:=pH^[k]^[49]+kc100(ro2)*pr;
 pH^[k]^[50]:=pH^[k]^[50]+kc102(ro2,cs2)*pr;
 pH^[k]^[51]:=pH^[k]^[51]+ks102(ro2,sn2)*pr;
 pH^[k]^[52]:=pH^[k]^[52]+kc104(ro2,cs4)*pr;
 pH^[k]^[53]:=pH^[k]^[53]+ks104(ro2,sn4)*pr;
 pH^[k]^[54]:=pH^[k]^[54]+kc111(ro,ro2,cs)*pr;
 pH^[k]^[55]:=pH^[k]^[55]+ks111(ro,ro2,sn)*pr;
 pH^[k]^[56]:=pH^[k]^[56]+kc113(ro,ro2,cs3)*pr;
 pH^[k]^[57]:=pH^[k]^[57]+ks113(ro,ro2,sn3)*pr;
 pH^[k]^[58]:=pH^[k]^[58]+kc120(ro2)*pr;
 pH^[k]^[59]:=pH^[k]^[59]+kc122(ro2,cs2)*pr;
 pH^[k]^[60]:=pH^[k]^[60]+ks122(ro2,sn2)*pr;
 pH^[k]^[61]:=pH^[k]^[61]+kc131(ro,ro2,cs)*pr;
 pH^[k]^[62]:=pH^[k]^[62]+ks131(ro,ro2,sn)*pr;
 pH^[k]^[63]:=pH^[k]^[63]+kc140(ro2)*pr;
 sv^[k]:=sv^[k]+wk*pr;
end;
```

```
var
 mm,w0: float;

procedure FormMatrix;

 procedure MatrixMod(x,y:float);
 var
  a,r: float;
 begin
   r:=sqrt(sqr(x)+sqr(y));
   if (((r+pix)>H0) or ((r-pix)<Hv)) then
    exit;
   ro:=r/H0;
   ro2:=sqr(ro);
   if (x=0) or (r=0) then
    begin
     if y>=0 then
      a:=0
     else
      a:=pi;
    end
   else
    a:=arccos(y/r);
   if (x<0) then
    a:=-a;
   cs:=cos(a);
   sn:=sin(a);
   cs2:=cos(a*2);
   sn2:=sin(a*2);
   cs3:=cos(a*3);
   sn3:=sin(a*3);
   cs4:=cos(a*4);
   sn4:=sin(a*4);
   cs5:=cos(a*5);
   sn5:=sin(a*5);
   cs6:=cos(a*6);
   sn6:=sin(a*6);
   cs7:=cos(a*7);
   sn7:=sin(a*7);
   Moderow(0,1);
   Moderow(1,kc11(ro,cs));
   Moderow(2,ks11(ro,sn));
   Moderow(3,kc20(ro2));
   Moderow(4,kc22(ro2,cs2));
   Moderow(5,ks22(ro2,sn2));
   Moderow(6,kc31(ro,ro2,cs));
   Moderow(7,ks31(ro,ro2,sn));
   Moderow(8,kc33(ro,ro2,cs3));
   Moderow(9,ks33(ro,ro2,sn3));
   Moderow(10,kc40(ro2));
   Moderow(11,kc42(ro2,cs2));
   Moderow(12,ks42(ro2,sn2));
   Moderow(13,kc44(ro2,cs4));
   Moderow(14,ks44(ro2,sn4));
```

```
   Moderow(15,kc51(ro,ro2,cs));
   Moderow(16,ks51(ro,ro2,sn));
   Moderow(17,kc53(ro,ro2,cs3));
   Moderow(18,ks53(ro,ro2,sn3));
   Moderow(19,kc55(ro,ro2,cs5));
   Moderow(20,ks55(ro,ro2,sn5));
   Moderow(21,kc60(ro2));
   Moderow(22,kc62(ro2,cs2));
   Moderow(23,ks62(ro2,sn2));
   Moderow(24,kc64(ro2,cs4));
   Moderow(25,ks64(ro2,sn4));
   Moderow(26,kc66(ro2,cs6));
   Moderow(27,ks66(ro2,sn6));
   Moderow(28,kc71(ro,ro2,cs));
   Moderow(29,ks71(ro,ro2,sn));
   Moderow(30,kc73(ro,ro2,cs3));
   Moderow(31,ks73(ro,ro2,sn3));
   Moderow(32,kc75(ro,ro2,cs5));
   Moderow(33,ks75(ro,ro2,sn5));
   Moderow(34,kc77(ro,ro2,cs7));
   Moderow(35,ks77(ro,ro2,sn7));
   Moderow(36,kc80(ro2));
   Moderow(37,kc82(ro2,cs2));
   Moderow(38,ks82(ro2,sn2));
   Moderow(39,kc84(ro2,cs4));
   Moderow(40,ks84(ro2,sn4));
   Moderow(41,kc86(ro2,cs6));
   Moderow(42,ks86(ro2,sn6));
   Moderow(43,kc91(ro,ro2,cs));
   Moderow(44,ks91(ro,ro2,sn));
   Moderow(45,kc93(ro,ro2,cs3));
   Moderow(46,ks93(ro,ro2,sn3));
   Moderow(47,kc95(ro,ro2,cs5));
   Moderow(48,ks95(ro,ro2,sn5));
   Moderow(49,kc100(ro2));
   Moderow(50,kc102(ro2,cs2));
   Moderow(51,ks102(ro2,sn2));
   Moderow(52,kc104(ro2,cs4));
   Moderow(53,ks104(ro2,sn4));
   Moderow(54,kc111(ro,ro2,cs));
   Moderow(55,ks111(ro,ro2,sn));
   Moderow(56,kc113(ro,ro2,cs3));
   Moderow(57,ks113(ro,ro2,sn3));
   Moderow(58,kc120(ro2));
   Moderow(59,kc122(ro2,cs2));
   Moderow(60,ks122(ro2,sn2));
   Moderow(61,kc131(ro,ro2,cs));
   Moderow(62,ks131(ro,ro2,sn));
   Moderow(63,kc140(ro2));
   inc(NPoints);
 end;

// формирование информационной матрицы для МНК
 var
```

```
  i,j,m,n: integer;
  ww,x0: float;
 begin
  for i:=0 to NpV-1 do
  begin
// коррекция начала BegPlsV и длины LenPlsV полос по уставке dend пользователя:
   m:=BegPlsV^[i]+dend;
    n:=LenPlsV^[i]-dend*2;
    if (m>0) and (n>0) then
// основной программный блок перевода кривизны полосы в фазу волновой ошибки:
   begin
     x0:=plsX0[i]; // абсцисса максимума в середине полосы
    for j:=m to m+n-1 do
      if abs(MarkPlsV[i]^[j].a)>0.2 then // 0.2 - отсекающий порог для
краёв полос
      begin
// вычисление искажения полосы и учет целочисленного приращения:
       wk:=w0+(MarkPlsV[i]^[j].x-x0)*mm+i;
// формирование информационной матрицы относительно центра ИГ:
       MatrixMod(MarkPlsV[i]^[j].x-xc,yc-MarkPlsV[i]^[j].y);
       end;
   end;
  end;
 end;

var
 i,j,o: integer;
 d0,di: float;
 csz: PVector;
 pp: PMatrix;
// структура волновой ошибки через полиномы Цернике:
{w=c00+c20+c40+c60+c80+c100+c120+c140+
   c11+c31+c51+c71+c91+c111+c131+ =sn1
   c22+c42+c62+c82+c102+c122+     =sn2
   c33+c53+c73+c93+c113+          =sn3
   c44+c64+c84+c104+              =sn4
   c55+c75+c95+                   =sn5
   c66+c86+                       =sn6
   c77                            =sn7
4:15,6:28,8:45,10:66,12:91,14:120 полиномов}
begin
 DimMatrix(pH,63,63);
 DimMatrix(pp,63,63);
 DimVector(sv,63);
 DimVector(csz,63);
 NPoints:=0;
 mm:=(nPolos-1)/2/H0; // цена полосы/мм
 w0:=mm*MarkPlsV[0]^[q].x; // начало отсчёта, «исходный нуль»
// формирование информационной матрицы
 FormMatrix;
// нормирование информационной матрицы
 for i:=0 to 63 do
 begin
  for j:=0 to 63 do
   pH^[i]^[j]:=pH^[i]^[j]/Npoints;
```

```
  sv^[i]:=sv^[i]/Npoints;
 end;
 copyMatrix(pp,pH,0,0,63,63);
// вычисление детерминанта информационной матрицы
 o:=GaussJordan(pp,0,63,63,d0);
 if o<>MAT_OK then
 begin
  showMessage('d0 invalid');
  exit;
 end;
// определения коэффициентов полиномов
 for i:=0 to 63 do
 begin
  copyMatrix(pp,pH,0,0,63,63);
  copyColfromVector(pp,sv,0,63,i);
  o:=GaussJordan(pp,0,63,63,di);
  if o<>MAT_OK then
  begin
   showMessage('d'+inttostr(i)+' invalid');
   exit;
  end;
  csz^[i]:=di/d0;
 end;
// коррекция коэффициентов с учётом весовых множителей:
 c00:=csz^[0];
 c11:=csz^[1]*2;
 s11:=csz^[2]*2;
 c20:=csz^[3]*sqrt(3);
 c22:=csz^[4]*sqrt(6);
 s22:=csz^[5]*sqrt(6);
 c31:=csz^[6]*sqrt(8);
 s31:=csz^[7]*sqrt(8);
 c33:=csz^[8]*sqrt(8);
 s33:=csz^[9]*sqrt(8);
 c40:=csz^[10]*sqrt(5);
 c42:=csz^[11]*sqrt(10);
 s42:=csz^[12]*sqrt(10);
 c44:=csz^[13]*sqrt(10);
 s44:=csz^[14]*sqrt(10);
 c51:=csz^[15]*sqrt(12);
 s51:=csz^[16]*sqrt(12);
 c53:=csz^[17]*sqrt(12);
 s53:=csz^[18]*sqrt(12);
 c55:=csz^[19]*sqrt(12);
 s55:=csz^[20]*sqrt(12);
 c60:=csz^[21]*sqrt(7);
 c62:=csz^[22]*sqrt(14);
 s62:=csz^[23]*sqrt(14);
 c64:=csz^[24]*sqrt(14);
 s64:=csz^[25]*sqrt(14);
 c66:=csz^[26]*sqrt(14);
 s66:=csz^[27]*sqrt(14);
 c71:=csz^[28]*4;
 s71:=csz^[29]*4;
 c73:=csz^[30]*4;
```

```
 s73:=csz^[31]*4;
 c75:=csz^[32]*4;
 s75:=csz^[33]*4;
 c77:=csz^[34]*4;
 s77:=csz^[35]*4;
 c80:=csz^[36]*3;
 c82:=csz^[37]*sqrt(18);
 s82:=csz^[38]*sqrt(18);
 c84:=csz^[39]*sqrt(18);
 s84:=csz^[40]*sqrt(18);
 c86:=csz^[41]*sqrt(18);
 s86:=csz^[42]*sqrt(18);
 c91:=csz^[43]*sqrt(20);
 s91:=csz^[44]*sqrt(20);
 c93:=csz^[45]*sqrt(20);
 s93:=csz^[46]*sqrt(20);
 c95:=csz^[47]*sqrt(20);
 s95:=csz^[48]*sqrt(20);
 c100:=csz^[49]*sqrt(11);
 c102:=csz^[50]*sqrt(22);
 s102:=csz^[51]*sqrt(22);
 c104:=csz^[52]*sqrt(22);
 s104:=csz^[53]*sqrt(22);
 c111:=csz^[54]*sqrt(24);
 s111:=csz^[55]*sqrt(24);
 c113:=csz^[56]*sqrt(24);
 s113:=csz^[57]*sqrt(24);
 c120:=csz^[58]*sqrt(13);
 c122:=csz^[59]*sqrt(26);
 s122:=csz^[60]*sqrt(26);
 c131:=csz^[61]*sqrt(28);
 s131:=csz^[62]*sqrt(28);
 c140:=csz^[63]*sqrt(15);
end;
 DelMatrix(pH,63,63);
 DelMatrix(pp,63,63);
 DelVector(sv,63);
 DelVector(csz,63);
end;
```

Найденные коэффициенты используются для вычисления зрачковой функции ОС:

```
procedure ZernPF;
var
 i,j,k,n,q,q2,q4,dend: integer;
 Hkn,Hkv,
 a,c,r,ro,ro2,cs,cs2,cs3,cs4,cs5,sn,sn2,sn3,sn4,sn5,cs6,cs7,kk,
 sn6,sn7,sw,wdf,wtt,ws,wk1,wk3,wa2,wa4,pf,mn,mx,da,iy,jz,y2: float;
begin
 q:=nSec;
 q2:=q div 2;
 q4:=q2 div 2;
 da:=H0/q4;
```

```
mx:=0;
mn:=0;
sw:=0;
n:=0;
dend:=paraCorrInt('pixEnd',frmMain.pixEnd_Edit.text);
Hkn:=H0-da*dend;
for i:=q4 to q4+q2-1 do
begin
 c:=q2-i;
 if c<0 then
  iy:=da*(c-0.5)
 else
  iy:=da*(c+0.5);
 y2:=sqr(iy);
 for j:=q4 to q4+q2-1 do
 begin
  c:=-(q2-j);
  if c<0 then
   jz:=da*(c-0.5)
  else
   jz:=da*(c+0.5);
  r:=sqrt(sqr(jz)+y2);
  if (r<=Hkn) and (r>=Hv) then
  begin
   ro:=r/H0;
   ro2:=sqr(ro);
   if (jz=0) or (r=0) then
   begin
    if iy>=0 then
     a:=0
    else
     a:=pi;
   end
  else
   a:=arccos(iy/r);
  if (jz<0) then
   a:=-a;
  cs:=cos(a);
  sn:=sin(a);
  cs2:=cos(a*2);
  sn2:=sin(a*2);
  cs3:=cos(a*3);
  sn3:=sin(a*3);
  cs4:=cos(a*4);
  sn4:=sin(a*4);
  cs5:=cos(a*5);
  sn5:=sin(a*5);
  cs6:=cos(a*6);
  sn6:=sin(a*6);
  cs7:=cos(a*7);
  sn7:=sin(a*7);
wdf:=C00+c20*kc20(ro2);
wtt:=c11*kc11(ro,cs)+s11*ks11(ro,sn);
ws:=c40*kc40(ro2)+c60*kc60(ro2)+c80*kc80(ro2)+
```

```
     c100*kc100(ro2)+c120*kc120(ro2)+c140*kc140(ro2);
 wk1:=c31*kc31(ro,ro2,cs)+s31*ks31(ro,ro2,sn)+
      c33*kc33(ro,ro2,cs3)+s33*ks33(ro,ro2,sn3)+
      c51*kc51(ro,ro2,cs)+s51*ks51(ro,ro2,sn)+
      c53*kc53(ro,ro2,cs3)+s53*ks53(ro,ro2,sn3)+
      c55*kc55(ro,ro2,cs5)+s55*ks55(ro,ro2,sn5);
 wk3:=c71*kc71(ro,ro2,cs)+s71*ks71(ro,ro2,sn)+
      c73*kc73(ro,ro2,cs3)+s73*ks73(ro,ro2,sn3)+
      c75*kc75(ro,ro2,cs5)+s75*ks75(ro,ro2,sn5)+
      c77*kc77(ro,ro2,cs7)+s77*ks77(ro,ro2,sn7)+
      c91*kc91(ro,ro2,cs)+s91*ks91(ro,ro2,sn)+
      c93*kc93(ro,ro2,cs3)+s93*ks93(ro,ro2,sn3)+
      c95*kc95(ro,ro2,cs5)+s95*ks95(ro,ro2,sn5)+
      c111*kc111(ro,ro2,cs)+s111*ks111(ro,ro2,sn)+
      c113*kc113(ro,ro2,cs3)+s113*ks113(ro,ro2,sn3)+
      c131*kc131(ro,ro2,cs)+s131*ks131(ro,ro2,sn);
 wa2:=c22*kc22(ro2,cs2)+s22*ks22(ro2,sn2)+
      c42*kc42(ro2,cs2)+s42*ks42(ro2,sn2)+
      c44*kc44(ro2,cs4)+s44*ks44(ro2,sn4)+
      c62*kc62(ro2,cs2)+s62*ks62(ro2,sn2)+
      c64*kc64(ro2,cs4)+s64*ks64(ro2,sn4)+
      c66*kc66(ro2,cs6)+s66*ks66(ro2,sn6);
 wa4:=c82*kc82(ro2,cs2)+s82*ks82(ro2,sn2)+
      c84*kc84(ro2,cs4)+s84*ks84(ro2,sn4)+
      c86*kc86(ro2,cs6)+s86*ks86(ro2,sn6)+
      c102*kc102(ro2,cs2)+s102*ks102(ro2,sn2)+
      c104*kc104(ro2,cs4)+s104*ks104(ro2,sn4);

// комплектации волновой ошибки из независимых составляющих:
     pf:=wtt+wdf+ws+wk1+wa2+wk3+wa4;
// удаление составляющей по уставкам программы пользователя:
     with frmMain do begin
       if dFAber_ChBx.State=cbChecked then
        pf:=pf-wdf;
       if SferAber_ChBx.State=cbChecked then
        pf:=pf-ws;
       if Kom1Aber_ChBx.State=cbChecked then
        pf:=pf-wk1;
       if Ast2Aber_ChBx.State=cbChecked then
        pf:=pf-wa2;
       if KomaVP_ChBx.State=cbChecked then
        pf:=pf-wk3;
       if AstVP_ChBx.State=cbChecked then
        pf:=pf-wa4;
       if TiltAber_ChBx.State=cbChecked then
        pf:=pf-wtt;
      end;
      PuplFn^[j+i*q]:=pf;
      sw:=sw+sqr(pf);
      inc(n);
      if abs(pf)>abs(mx) then
       mx:=pf;
      if abs(pf)<abs(mn) then
       mn:=pf;
    end;
```

```
  end;
 end;
 sw:=sqrt(sw/n);
 maxW:=mx-mn;
 if maxW=0 then
  maxW:=1;
 with frmMain do
 begin
  dW_Ed.Text:=FormatFloat('0.000',sw*Lambda/2);
  Wmx_Ed.Text:=FormatFloat('0.000',maxW*Lambda/2);
 end;
end;
```

III. ПРИЛОЖЕНИЯ

1. Литература

1. Апенко М. Прикладная оптика / М. Апенко, А. Дубовик. – М.: Наука, 1971.

2. Борн М. Основы оптики / М. Борн, Э. Вольф. – М.: Наука, 1975.

3. Гвоздева Н.П. Теория оптических систем и оптические измерения / Н.П. Гвоздева, К.И. Коркина. – М.: Машиностроение, 1981.

4. Чуриловский В.Н. Теория оптических приборов / В.Н. Чуриловский. – М.-Л.: Машиностроение, 1966.

5. Чуриловский В.Н. Теория хроматизма и аберраций 3-го порядка / В.Н. Чуриловский. – Л.: Машиностроение, 1968.

6. Слюсарев Г.Г. Расчет оптических систем / Г.Г. Слюсарев. – Л.: Машиностроение, 1975.

7. Слюсарев Г.Г. О возможном и невозможном в оптике / Г.Г. Слюсарев. – Л.: Машиностроение, 1953.

8. Русинов М. Техническая оптика / М. Русинов. – Л.: Машиностроение, 1979.

9. Русинов М.М. Несферические поверхности в оптике / М.М. Русинов. – М.: Недра, 1973.

10. Проектирование оптических систем. Под ред Р. Шеннона и Дж. Вайанта. – М.: Мир, 1983.

11. Джерард А. Введение в матричную оптику / А. Джерард, Дж.Н. Берг. – М.: Мир, 1978.

12. Леонова В.Б. Автоматизация расчётов оптических систем / В.Б. Леонова. – М.: Маширотроение, 1970.

13. Михельсон Н.Н. Оптические телескопы / Н.Н. Михельсон. – М.: Наука, 1976.

14. Теребиж В.Ю. Современные оптические телескопы / В.Ю. Теребиж. – М.: ФизматГИз, 2005.

15. Волосов Д.С. Фотографическая оптика / Д.С. Волосов. – М.: Искусство, 1978.

16. Каширин В.И. Универсальная асферическая поверхность / В.И. Каширин. Деп. в ВИНИТИ 26 дек. 1986, № 8358-84; РЖФиз. 1985. № 3. 3Л 570.

17. Попов Г.М. Современная астрономическая оптика / Г.М. Попов. – М.: Наука, 1988.

18. Сокольский М.Н. Допуски и качества оптического изображения / М.Н. Сокольский. - Ленинград; Машиностроение, 1989.

19. Каширин В.И. Основы формообразования оптических поверхностей : курс лекций / В.И. Каширин. – Екатеринбург: ГОУ ВПО УГТУ – УПИ, 2006.

20. Программа ROS#: разработка оптических систем. Справочное руководство к программе РОС / Екатеринбург, 2015.

21. Грамматин А.П. Компьютерное моделирование при изучении дмсциплин, связанных с расчётом оптических систем: методические указания к лабораторным работам / А.П. Грамматин, Г.Э. Романова, Е.А. Цыганок. – СПб, 2011.

22. Грамматин А.П. Методы проектирования. Автоматизация проектирования оптических систем: методические указания к лабораторным работам / А.П. Грамматин, Г.Э. Романова. – СПб: СПбГУ ИТМО, 2008.

23. Грамматин А.П. Расчёт и автоматизация проектирования оптических систем: учебное пособие / А.П. Грамматин, Г.Э. Романова, О.Н. Балаценко. – СПб: НИУ ИТМО, 2013.

2. ROS#: программа РОС для пользователя.

Режим доступа: https://yadi.sk/d/vviS6jkzxqKq4

3. prjROS#: проект Delphi программы РОС с исходным программным кодом на языке Паскаль.

Режим доступа: https://yadi.sk/d/MWrp88SiEGk-CA

4. ostPRJ: проект Delphi программы ОСТ с исходным программным кодом на языке Паскаль.

Режим доступа: https://yadi.sk/d/1CS-Wnihbgser

5. ОСТ: программа «Тестирование Оптических Систем» для пользователя.

Режим доступа: https://yadi.sk/d/8X5RQIGEcEmUX

6. Delphi7Lite: дистрибутив среды программирования фирмы Borland.

Режим доступа: https://yadi.sk/d/N4u6GIV23pgYs