GF User Guide

George Y. Panasyuk

Department of Bioengineering, University of Pennsylvania, Philadelphia, PA 19104

E-mail: `georgey@seas.upenn.edu`

John C. Schotland

Department of Bioengineering and Graduate Group in Applied Mathematics and Computational Science, University of Pennsylvania, Philadelphia, PA 19104

E-mail: `schotland@seas.upenn.edu`

Vadim A. Markel

Departments of Radiology and Bioengineering, University of Pennsylvania, Philadelphia, PA 19104

E-mail: `vmarkel@mail.med.upenn.edu`

Abstract.

This *User Guide* describes a package of Fortran-77 codes which compute the components of the frequency-domain electromagnetic Green's tensor in the half-space geometry. The Green's tensor gives the electric field at the point $\mathbf{r}$ due to a monochromatically oscillating dipole at the point $\mathbf{r}'$.

The package is deposited at the *J.Phys.A* web site as a supplemental material to the following paper: G.Y.panasyuk, J.C.Schotland and V.A.Markel, "Short-distance expansion for the electromagnetic half-space Green's tensor: general results and an application to radiative lifetime computations".

The authors welcome questions, comments or bug reports at the following e-mail address: `vmarkel@mail.med.upenn.edu`. Please check the following URL for updates and additional information: `http://whale.upenn.edu/CODES/`.

1. Important preliminary notes

Before starting to use the programs, please familiarize yourself with the following important points which are listed below in no particular order.

- The Green's tensor is computed in the special reference frame shown in Fig. 1 of the paper.
- All programs use the same input parameter file `GF.par`. The parameters can be used to tune the execution in many different ways. See Sec. 4 for details.
- Information about execution progress, conflicting or invalid parameters is printed on the computer display. Do not ignore this information; it is provided for a reason. If invalid parameters are read from GF.par, execution will stop with a brief explanation. Correct the error in GF.par but do not attempt to edit the programs to override the checks. In some instances, the programs would want to alert you of something. Then the execution will pause rather then stop. You can resume execution by typing `go` or pressing the `ENTER` key, depending on your Fortran compiler.
- Output data are written to files and not printed on the display. There are two types of files: those whose name start with a `G` and those whose name start with a `K`. The first contain the components of the Green's tensor and the second contain the components of the expansion tensor coefficients $\mathcal{K}^{(l)}$.
- The Green's tensor in the output files is dimensionalized by multiplying it by the factor k_1^{-3}. Here $k_1 = \omega\sqrt{\epsilon_1}/c = 2\pi\sqrt{\epsilon_1}/\lambda$. The permittivity of the upper half-space where the source and the point of observation are located, ϵ_1 is a positive constant, possibly, unity.
- A single run of any of the programs in this package will perform the computations for fixed coordinates of the source and the point of observation and for multiple (possibly, just one) wavelengths. All output data are viewed as functions of the free-space wavelength $\lambda = 2\pi c/omega$. This quantity is printed in the first column of all output files.
- All input and output quantities which have the dimensionality of length are assumed to be in nanometers.
- All programs are written in Fortran-77 and have been tested with the GNU's g77 compiler.

2. List of files

`GF_User_Guide.pdf`	This *Guide*
`GF.par`	A file containing input parameters for all programs.
`GF_num_inb.f`	Numerical computation of the Green's tensor using intrinsic Bessel functions BESJ0 and BESJ1.
`GF_num_exb.f`	Numerical computation of the Green's tensor using an external subroutine to compute the Bessel functions.
`GF_sde_asub.f`	Computation of the Green's tensor using expansion (28) up to third order (see paper). Applicable to all types of substrates.
`GF_sde_tnds.f`	Computation of the Green's tensor using expansion (28) up to seventh order (see paper). Applicable to nonabsorbing, nondispersive substrates. (Wavelength-independent purely real, positive permittivity ϵ_2.)
`GF_sde_tdss.f`	Computation of the Green's tensor using expansion (28) up to seventh order (see paper). Applicable to nonabsorbing but dispersive substrates. (Purely real, positive, but wavelength-dependent permittivity ϵ_2.) Requires additional programming to define the function $\epsilon_2(\lambda)$.
`GF_k2g`	Uses tensor expansion coefficients $\mathcal{K}$ previously written by programs `GF_sde_*.f` to compute the Green's tensor $\mathcal{G}^R$ according to the expansion (28).

Table of acronyms used in the filenames

`GF`	Greens Function
`num`	Numerical
`sde`	Short-distance expansion (formula (28))
`asub`	Absorbing substrate
`tsub`	Transparent substrate (used in output files only)
`tnds`	Transparent nondispersive substrate
`tdss`	Transparent dispersive substrate
`k2g`	Computation of $\mathcal{G}^R$ using precomputed $\mathcal{K}$

3. Description of programs

3.1. `GF_num_inb.f` *and* `GF_num_exb.f`

These two programs are very similar and compute the same quantities: the dimensionalized reflected part of the Green's tensor, $k_1^{-3}\mathcal{G}^R$, and the dimensionalized total Green's tensor, $k_1^{-3}G$ defined in Eqs. (5-8),(16), by numerical integration according to the Simpson's rule. See additional notes on integration in Sec. 6.

Recall that $k_1 = \omega n_1/c$ is the wave number in the upper (transparent) half-space where $n_1 = \sqrt{\epsilon_1} > 0$. We can also write $k_1 = 2\pi n_1/\lambda$ where λ is the wavelength which corresponds to the frequency ω in vacuum.

It is important to remember that this and all other programs compute the Green's tensor as a function of the free-space wavelength λ. This quantity is printed in the first column of all output files. However, the wavelength in the upper half-space (where the source and the point of observation are located) may be different if the parameter `eps_1` is chosen to be different from unity.

A single run of either of these two programs will perform computations for fixed source and detector positions and for multiple (possibly, just one) wavelengths λ.

The difference between the two programs is the following. The program `GF_num_inb.f` makes use of the intrinsic Bessel functions BESJ0 and BESJ1 while `GF_num_exb.f` calls an external subroutine which is adapted (with some modifications) from the numerical library written by S.Zhang and J.Jin; see *Computation of Special Functions*, Wiley, 1996, Sec.5.2. There is a very minor numerical discrepancy between the results of these two programs (in sixth or seventh significant figure). It is not really possible to tell which version is more accurate. In all tests, the numerical differences were very insignificant. However, the program `GF_num_exb.f` runs approximately twice faster because the external subroutine computes the Bessel functions of the zeroth and the first order in one call, which is more efficient than computing these two functions separately.

The only constraint on the substrate permittivity ϵ_2 that is placed by this program is that its imaginary part must be non-negative. A purely real permittivity is OK. There are several ways to define the permittivity. This is explained in more detail in Sec. 5.

Output. The output of these two programs is sent to the files named:

`GRxx_num`, `GRyy_num`, `GRzz_num`, `GRxz_num`, `GRzx_num`

`GTxx_num`, `GTyy_num`, `GTzz_num`, `GTxz_num`, `GTzx_num`

The files whose names start with `GR` contain the reflected part of the Green's tensor while the files whose names start with `GT` contain the total Greens tensor. The indices `xx`, `yy`, `zz`, `xz` and `zx` are self-explanatory. They label the components of the Green's tensor in the special reference frame shown in Fig. 1 of the paper. Each line of the output files contains three numbers: the free space wavelength, the real part and the imaginary part of the Green's tensor component which is specified in the filename. There may be an arbitrary number of lines in each file, as specified by the input parameter `nl`. The wavelength can be sampled either linearly or using a logarithmic scale, as specified by the input parameter `lambda_scale`.

Note that zx and xz components of the Green's tensor satisfy $\mathcal{G}^R_{xz} = -\mathcal{G}^R_{zx}$q. Therefore, the data in files `GRxz_num` and `GRzx_num` are, in fact, redundant: they only differ by sign. The total Green's tensor, however, does not have this symmetry. Therefore, the files `GTxz_num`, `GTzx_num` are not redundant and contain mathematically-independent data.

3.2. `GF_sde_asub.f`

This program computes the expansion coefficients $\mathcal{K}^{(l)}$ for $l = 0, 1, 2, 3$. It also computes the reflected part of dimensionalized Green's tensor, $k_1^{-3}\mathcal{G}^R$, by performing summation in Eq.(28). The program adds all terms in this expansion with $l \leq$ `nord` ≤ 3, where `nord` is an input parameter.

Only the reflected part of the Green's tensor is computed. Wondering why? See the IAQ below.

Just like the numerical integration programs, this program is applicable to substrate of the most general type, including absorbing and strongly dispersive substrates. See Sec. 5 for details of specifying the substrate permittivity ϵ_2.

Note that the expansion (28) given in the paper reads for the dimensionalized Green's tensor:

$$k_1^{-3}\mathcal{G}^R = \sum_{l=0}^{\infty}(k_1\mathcal{L})^{l-3}\mathcal{K}^{(l)} \ .$$

Components of $\mathcal{G}^R$ written to output files are computed according to this formula. Note that same is true for all programs `GF_sde_*.f`.

Output. The output of `GF_sde_asub.f` is sent to the files named

`Kxx_asub`, `Kyy_asub`, `Kzz_asub`, `Kxz_asub`

`GRxx_asub`, `GRyy_asub`, `GRzz_asub`, `GRxz_asub`

Files `Kzx_asub` and `GRzx_asub` are not created due to the symmetry $\mathcal{K}_{xz} = -\mathcal{K}_{zx}$, $\mathcal{G}^R_{xz} = -\mathcal{G}^R_{zx}$ (the total Green's tensor does not have this symmetry but it is not computed by this program).

Each line in the files whose names start with `K` contains seven numbers. The first number is the wavelength, λ. The next two numbers are the real and the imaginary part of the tensor component of $\mathcal{K}^{(0)}$ which corresponds to the letters in the filename. Then, in the same manner, follow the real and imaginary parts of $\mathcal{K}^{(2)}$ and of $\mathcal{K}^{(3)}$. $\mathcal{K}^{(1)}$ is not given because it is identically zero. There may be arbitrarily many lines.

The files `GRxx_asub`, `GRyy_asub`, `GRzz_asub`, `GRxz_asub` are completely similar to the files `GRxx_num`, `GRyy_num`, `GRzz_num`, `GRxz_num`, except that the suffix `asub` is used to emphasize that these data are obtained by a different method (and to avoid overwriting previously created output files).

It is important to remember that the Green's tensor whose components are given in `GRxx_asub`, `GRyy_asub`, `GRzz_asub`, `GRxz_asub` is computed to the order specified by the input parameter `nord`. However, the files `Kxx_asub`, `Kyy_asub`, `Kzz_asub`, `Kxz_asub` contain all the expansion coefficients $\mathcal{K}^{(l)}$ for $l \leq 3$, irrespectively of `nord`, even if some of these coefficients have not been used to compute $\mathcal{G}^R$. These output files can later be used as input for the program `GF_k2g.f` (see Sec. 7).

3.3. `GF_sde_tnds.f`

This program is similar to `GF_sde_asub.f` but differs from the latter in the following two respects:

(i) The program will only work with purely real, positive and wavelength-independent (nondispersive) permittivity of the substrate, ϵ_2. If you give a nonzero imaginary part of ϵ_2 in the `GF.par` file (the input parameter `eps_2_i`), it will be ignored.
It is possible to run this program several times for different single wavelengths specifying each time a different permittivity of the substrate (input parameter `eps_2_r`). A more direct way to account for dispersion is, however, to use `GF_sde_tdss.f`.

(ii) The program computes the expansion of the imaginary part of the dimensionalized reflected Green's tensor $\mathcal{G}^R$ up to the seventh order. That is, imaginary parts of all terms in Eq.(28) with $l \leq$ `nord` ≤ 7 are summed up. The real part is still computed to no higher than third order.

Output. The output of `GF_sde_tnds.f` is sent to the files named

`Kxx_tsub`, `Kyy_tsub`, `Kzz_tsub`, `Kxz_tsub`

`GRxx_tsub`, `GRyy_tsub`, `GRzz_tsub`, `GRxz_tsub`

Files `Kzz_tsub` and `GRzz_tsub` are not created due to the symmetry $\mathcal{K}_{xz} = -\mathcal{K}_{zx}$, $\mathcal{G}^R_{xz} = -\mathcal{G}^R_{zx}$ (the total Green's tensor does not have this symmetry but it is not computed by this program).

The only difference between the output of `GF_sde_tnds.f` and `GF_sde_asub.f` is that the files `Kxx_tsub`, `Kyy_tsub`, `Kzz_tsub`, `Kxz_tsub` contain now 15 rather then 7 numbers. The first number in each line is still the wavelength. The go the real and imaginary parts of $\mathcal{K}^{(0)}$, $\mathcal{K}^{(2)}$, $\mathcal{K}^{(3)}$, $\mathcal{K}^{(4)}$, $\mathcal{K}^{(5)}$, $\mathcal{K}^{(6)}$, $\mathcal{K}^{(7)}$ (14 numbers total). Note however that the real parts of all $\mathcal{K}^{(l)}$ with $l > 3$ are set to zero. Do not be surprised to see columns of zeros in files `Kxx_tsub`, etc. Wondering why? See IAQ.

3.4. `GF_sde_tdss.f`

This program is completely similar `GF_sde_tnds.f` except that it allows the substrate to be dispersive. That is, its permittivity ϵ_2 can depend on the wavelength. However, the user will need to program a specific formula for this dependence. Compile and run the program, and it will print out instructions on how to proceed.

Output of `GF_sde_tdss.f` is completely analogous to that of `GF_sde_tnds.f`.

3.5. `GF_k2g.f`

This program takes the output of the programs `GF_sde_asub.f` AND/OR (`GF_sde_tnds.f` OR `GF_sde_tnds.f`) and computes the reflected part of the dimensionalized Green's tensor, $k_1^{-3}\mathcal{G}^R$, according to the equation on p.5 of this *Guide* (this is equivalent to using Eq. (28) of the paper).

The behavior of `GF_k2g.f` depends on the input parameter `nord`. See Sec. 7 for more details.

Output. `GF_k2g.f` sends its output to files named

GRxx_k2g, GRyy_k2g, GRzz_k2g, GRxz_k2g

These files are completely analogous to the output files of the program `GF_num_exb.f`; see the description above.

4. Input

The same input file `GF.par` is used by all programs. However, not every parameter in this file is used by every program. The first two lines in `GF.par` are for informative purposes and are ignored on input. Deleting or editing these lines may cause the programs to complain about input or crash.

The remaining 15 lines contain the following input parameters:

1: `key` [Type: `INTEGER*4`, allowed values: `0,1,2`]

This parameter determines how the permittivity of the substrate, ϵ_2, is computed. If `key=0`, the substrate is assumed to be metallic and the Drude formula is used for its permittivity. In this case, parameters `lambda_p`, `gd` and `eps0` are also used to determine ϵ_2. If `key=1`, ϵ_2 is taken to be wavelength-independent with real and imaginary parts specified by the parameters `eps_2_r` and `eps_2_i`.

Note that the `key`=0 option implies that ϵ_2 is a function of the wavelength while `key`=1 option implies that ϵ_2 is a wavelength-independent constant, at least during a single run of any of the programs described in this *Guide*. It is, of course, possible to change the parameters `eps_2_r` and `eps_2_i` between consecutive runs.

The option `key=2` is for user-defined permittivity. It requires (very minimal) additional programming. Set `key`=2, run the program you wish to use and it will print instructions on how to proceed.

Not all programs allow all values of key and some programs will ignore the imaginary part given by the input parameter eps_2_i. Read the runtime messages carefully.

2: nord [Type: INTEGER*4, allowed values: $-7 \leq$ nord ≤ 7

This parameter tells to which order to compute $\mathcal{G}^R$. Numerical integration programs GF_num_*.f ignore this parameter. The programs GF_sde_*.f ignore the sign of this parameter. The program GF_sde_asub.f can not compute the expansion to orders higher than the third; therefore, if you feed it nord>3, it will complain and set (with your permission) nord=3.

The only program that uses the sign of nord is GF_k2g.f. If on input nord<-3, it will attempt to use the coefficients from files K**_asub for orders $l = 0, 1, 2, 3$ and from K*_tsub, for $l > 3$. This is advantageous when ϵ_2 has a small but non-negligible imaginary part and you want to take it into account. See more details in Sec. 7.

3: output_prec [Type: CHARACTER*4, allowed values: S,s,D,d]

This option controls the number of digits in the numbers written to output files. All computations are done in double precision, but the output can be written with single precision to simplify viewing and processing of files. Obviously, 'S' or 's' is for single precision output and 'D' or 'd' is for double precision.

Note that the wavelength is always written with single precision, irrespectively of the value of output_prec.

4: lambda_scale [Type: CHARACTER*4, allowed values: LIN, lin, LOG, log]

All programs perform calculations for multiple wavelength starting from the minimum value given by lambda_min and ending with the maximum value given by lambda_max. The number of wavelength samples is given by nl. The parameter lambda_scale controls whether the sampling is linear or logarithmic. The latter option is useful if the ratio lambda_max/lambda_min is much larger than unity and you wish to plot the results in a logarithmic scale.

5: lambda_min, lambda_max, nl [Type: REAL*8, REAL*8, INTEGER*4]

The minimum and the maximum wavelengths (in nanometers), and the total number of wavelength samples to be used in computations. The samples are chosen to include both lambda_min and lambda_max. The wavelength can be sampled in both linear and logarithmic scales. See the description of lambda_scale.

The programs will warn you if you give invalid or conflicting values for these parameters. Note that if nl=1, only one wavelength, namely, lambda_min will be used. Also, if lambda_min = lambda_max, the parameter nl is reset to unity.

6: z1, z2, rho [Type: REAL*8, allowed values: must be positive]

z1 and z2 are the heights of the source and the point of observation above the substrate. rho is the lateral distance between these two points (in nanometers). More specifically, this is the distance between the projections of the source and the point of detection onto the plane $z = 0$.

7: `eps_2_r` and `eps_2_i` [Type: `REAL*8`, allowed values: `eps_2_i` ≥ 0]

If `key`=1, these parameters are interpreted as real and imaginary parts of the substrate permittivity ϵ_2. Unlike in the case `key=0`, it will be assumed to be wavelength-independent.

The codes `GF_sde_t*.f` will ignore the parameter `eps_2_i` and not allow `eps_2_r` to be negative or zero.

8: `lambda_p` [Type: `REAL*8`, allowed values: must be positive]

If `key`=0, this variable is interpreted as the wavelength at the plasma frequency in nanometers and is used in the Drude formula for ϵ_2. See Sec. 5.

9: `gd` [Type: `REAL*8`, allowed values: must be positive]

If `key`=0, this variable is interpreted as the dimensionless ratio of the Drude relaxation constant, γ, to the plasma frequency, ω_p. For good conductors, this ratio is significantly less than unity. See Sec. 5.

Although the programs will allow arbitrary positive values of this parameter, numerical integration will require a very large number of points if `gd` $\lesssim 0.001$. Numerical integration programs will issue a warning is such a small value is given on input. However, the smallest practically attainable value of this constant is about 0.002 (for silver).

10: `eps0` [Type: `REAL*8`, allowed values: no restrictions]

If `key`=0, this variable is interpreted as the contribution to the substrate permittivity, ϵ_2 due to the inter-zone transitions (the non-Drudean part). For noble metals such as silver or gold, `eps0` ≈ 5. In the pure Drude model, `eps0` ≈ 1. There are currently no restrictions imposed on `eps0`. See Sec. 5.

11: `eps_1` [Type: `REAL*8`, allowed values: must be positive]

This is the permittivity of the upper half-space where the source and the point of observation are located. If this half space is vacuum, set `eps_1`=1.

12-14: `N1`, `N2` and `N3` [Type: `INTEGER*8`, allowed values: `N`$* \geq 10$]

These are numbers of discretization points used in numerical integration by programs `GF_num_*.f`. In practice, should not be smaller than 10^3. generally, much larger numbers are required for metal substrates than for dielectric substrates. See Sec. 6 for details.

It is a very good idea to use the same values for `N1`, `N2` and `N3`, at least initially. The possibility to use different values for these parameters is provided for fine-tuning, but should be used with caution.

In the Simpson integration scheme, the numbers `N1`, `N2` and `N3` must be even. If you give an odd number for any of these three parameters (why would you want to do that?), it will be converted to an even number by subtracting unity.

The variable `N1`, `N2` and `N3` are declared as INTEGER*8 in case someone wants to run integration with more than 2147483647 points, although in practice, this will hardly be ever necessary.

15: `y_max` and `div` [Type: `REAL*8`, allowed values: must be positive]

These parameters are used by `GF_num_*.f` programs for numerical integration.

The parameter `y_max` should be large; it's a good idea to start with 10^4 and check convergence by increasing this number in multiples of 10.

The parameter `div` in most case should be equal to 10. The capability to change this parameter is provided but should be used with caution. See Sec. 6.

5. Defining the substrate permittivity

5.1. `key`*=0.*

If the option `key` $= 0$ is used, the substrate is assumed to be metal and its permittivity is determined by the formula

$$\epsilon_2 = \epsilon_0 - \frac{\omega_p^2}{\omega(\omega + i\gamma)} = \epsilon_0 - \frac{1}{(\lambda_p/\lambda)[(\lambda_p/\lambda) + i(\gamma/\omega_p)]} \ .$$

In the above formula, ϵ_0 is specified by the parameter `eps0`. For purely Drudean metals, ϵ_0 should be unity. A deviation of this constant from unity is usually attributed to interzone transitions.

The constant $\lambda_p = 2\pi c/\omega_p$ is the wavelength at the plasma frequency, in nanometers. It is specified by the parameter `lambda_p`.

The free-space wavelength $\lambda = 2\pi c/\omega$ is sampled by the programs. The number of samples is determined by the parameter `nl` and the interval and scale of sampling by parameters `lambda_min`, `lambda_max`, `lambda_scale`.

Experimental values of the input parameters are listed below for several metals

Metal	`lambda_p` $= \lambda_p$ [nm]	`gd` $= \gamma/\omega_p$
Al	84	0.0054
W	207	0.0089
Pb	161	0.023
Cu	157	0.0018
Ag	136	0.0019
Au	138	0.0027

5.2. `key`*=1.*

If this option is chosen, then ϵ_2 is determined by reading the parameters `eps_2_r` and `eps_2_i`. The first is interpreted as the real part of ϵ_2. The second parameter may or may not be interpreted as the imaginary part. More specifically, the programs `GF_num_*.f` and `GF_sde_asub.f` will read the parameter `eps_2_i` and interpret it as the imaginary part of ϵ_2. However, the programs `GF_sde_t*.f` do not allow complex values of ϵ_2. These programs will simply ignore `eps_2_i`.

If `key=1` is selected, each program will assume that ϵ_2 is independent of λ (during a single run, of course). If you wish to use wavelength-dependent ϵ_2 of a type different from the Drudean-type formula which is used if `key=0`, use `key=2` and program your own formula.

5.3. `key`*=2*

This option is used to allow the user to program his/her own formula for ϵ_2.

To do so, open the program you wish to use and find the following piece of code:

```
if(key .eq. 2) then
   ! Enter your own formula for eps_2 here
```

or

```
else if(key .eq. 2) then
   ! Enter your own formula for eps_2 here
```

This piece of code is within a loop in which the variable `lambda` gives the free-space wavelength in nanometers. Use it to program your own formula. You can define any additional constants you wish, but remember that all variables must be declared.

6. Details of numerical integration

Let us multiply Eq. (17) from the paper by k_1^{-3} and write the result here:

$$k_1^{-3}\mathcal{G}^R(\mathcal{R},\mathcal{Z}) = \frac{1}{k_1}\int_0^\infty \frac{q\,dq}{\varkappa_1(q)} \exp\left[-\varkappa_1(q)\mathcal{Z}\right] F(q,\mathcal{R}) \ .$$

Here $\kappa_1(q) = \sqrt{q^2 - k_1^2}$ and the function $F(q,\mathcal{R})$ is defined in the paper.

We then consider the intervals $q \in (0, k_1)$ and $q \in (k_1, \infty)$ separately (recall that k_1 is a purely real, positive number). We then make the change of variables $\sqrt{1-(q/k_1)^2} = y$ in the first interval and $\sqrt{(q/k_1)^2 - 1} = y$ in the second interval, which yields

$$k_1^{-3}\mathcal{G}^R(\mathcal{R},\mathcal{Z}) = I_1 + I' \ ,$$

where

$$I_1 = -i\int_0^1 \exp\left[i(k_1\mathcal{Z})y\right] F\left(k_1\sqrt{1-y^2},\mathcal{R}\right) dy \ ,$$

$$I' = \int_0^\infty \exp\left[-(k_1\mathcal{Z})y\right] F\left(k_1\sqrt{1+y^2},\mathcal{R}\right) dy \ .$$

The integrand in the second of these integrals changes rapidly when y is small and slow when y is large. It is, therefore, necessary to break the integral I' in two parts, one over the interval $(0, d)$ and the other over the inter over the interval (d, ∞). Numerically, we can not handle infinities, so that the last integral is evaluated over the interval $(d, y_{\max})$. The values of d and $y_{\max}$ are specified by the input variables `div` and `y_max`, respectively.

We thus have:

$$k_1^{-3}\mathcal{G}^R(\mathcal{R}, \mathcal{Z}) \approx I_1 + I_2 + I_3 \ ,$$

where I_1 is given above and I_2, I_3 are

$$I_2 = \int_0^d \exp\left[-(k_1\mathcal{Z})y\right] F\left(k_1\sqrt{1+y^2}, \mathcal{R}\right) dy \ ,$$
$$I_3 = \int_d^{y_{\max}} \exp\left[-(k_1\mathcal{Z})y\right] F\left(k_1\sqrt{1+y^2}, \mathcal{R}\right) dy \ .$$

Now, each of the integrals I_1, I_2 and I_3 is computed by discretization according to the Simpson Rule. The number of discretization points for the integrals is specified by the input variables `N1`, `N2` and `N3`, respectively.

Examples of numerical convergence are shown in Fig. 1 for a silver substrate. Here the input file GF.par was the following:

```
Do not edit:     !line! variable                     ! type     !
Do not edit:     ----------------------------------------------
0                ! 1  ! key                          ! I4       !
2                ! 2  ! nord                         ! I4       !
S                ! 3  ! output_prec                  ! CHAR4    !
LIN              ! 4  ! lambda_scale                 ! CHAR4    !
400.,2000.,300   ! 5  ! lambda_min, lambda_max, nl   ! R8,R8,I4 !
40.,40.,40.      ! 6  ! z1, z2, rho                  ! R8,R8,I4 !
2.5,0.01         ! 7  ! eps_2_r, eps_2_i             ! R8,R8    !
136.1            ! 8  ! lambda_p                     ! R8       !
0.0019           ! 9  ! gd                           ! R8       !
5.0              ! 10 ! eps0                         ! R8       !
1.0              ! 11 ! eps_1                        ! R8       !
<VARIABLE>       ! 12 ! N1                           ! I8       !
<VARIABLE>       ! 13 ! N2                           ! I8       !
<VARIABLE>       ! 14 ! N3                           ! I8       !
<VARIABLE>,10.   ! 15 ! y_max, div                   ! R8,R8    !
```

and the parameters `N1`, `N2` and `N3` were taken to be the same and equal to N.

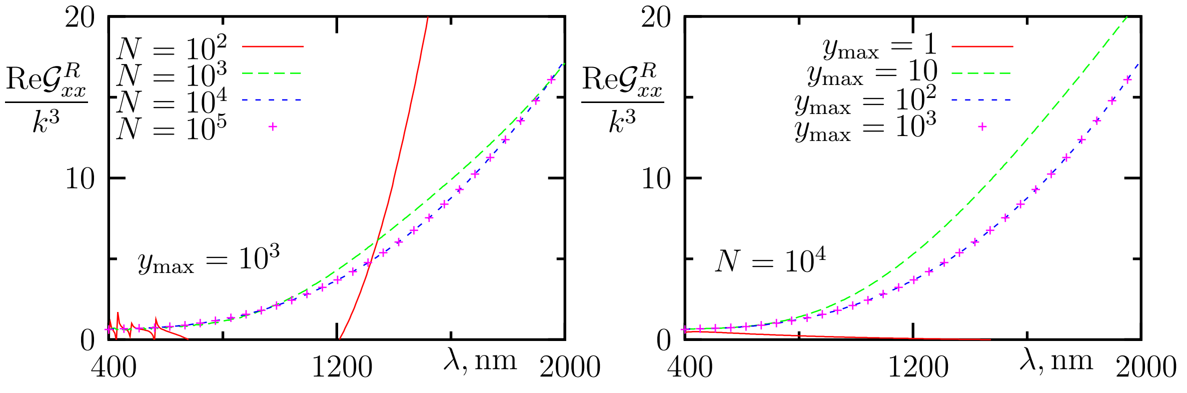

Figure 1. Illustration of numerical convergence of integrals for a silver substrate (xx-component); $k = k_1 = 2\pi c/\lambda$.

7. Using `GF_k2g.f`

Imagine that the permittivity of the substrate is constant but has a non-negligible imaginary part. For example: $\epsilon_2 = 2.5 + 0.1i$. The program `GF_sde_asub.f` can only compute the short-distance expansion to third order. The code `GF_sde_tnds.f` can compute the expansion to seventh order, but it will neglect the imaginary part of ϵ_2. What should one do in this situation?

The answer is rather simple. One should compute the expansion coefficients to third order using `GF_sde_asub.f` for $\epsilon_2 = 2.5 + 0.1i$. Then one should compute the imaginary parts of the expansion coefficients to seventh order using `GF_sde_tnds.f`. Then one should use `GF_k2g.f` with `nord` $= -7$. This will take the expansion coefficients which account for the small imaginary part of ϵ_2 to third order and the coefficients which do not account for this small imaginary part for orders 4 to 7. Since radiative effects are still dominating for this substrate, this is better than not using the higher order corrections at all or neglecting the imaginary part in all orders. This is illustrated in Fig. 2. The input file used in calculations (used by all three programs) is given below. The example illustrates both the usefulness of higher-order corrections and the usefulness of the `GF_k2g.f` program.

Note that `GF_k2g.f` must run the last. Otherwise, the order of execution does not matter. Goodbye.

```
Do not edit:    !line! variable                    ! type     !
Do not edit:    -------------------------------------------------
1               ! 1  ! key                         ! I4       !
-7              ! 2  ! nord                        ! I4       !
S               ! 3  ! output_prec                 ! CHAR4    !
LIN             ! 4  ! lambda_scale                ! CHAR4    !
400.,2000.,300  ! 5  ! lambda_min, lambda_max, nl  ! R8,R8,I4 !
40.,40.,40.     ! 6  ! z1, z2, rho                 ! R8,R8,I4 !
```

```
2.5,0.01        ! 7  ! eps_2_r, eps_2_i          ! R8,R8    !
136.1           ! 8  ! lambda_p                  ! R8       !
0.0019          ! 9  ! gd                        ! R8       !
5.0             ! 10 ! eps0                      ! R8       !
1.0             ! 11 ! eps_1                     ! R8       !
10000           ! 12 ! N1                        ! I8       !
10000           ! 13 ! N2                        ! I8       !
10000           ! 14 ! N3                        ! I8       !
1.0d3,10.0      ! 15 ! y_max, div                ! R8,R8    !
```

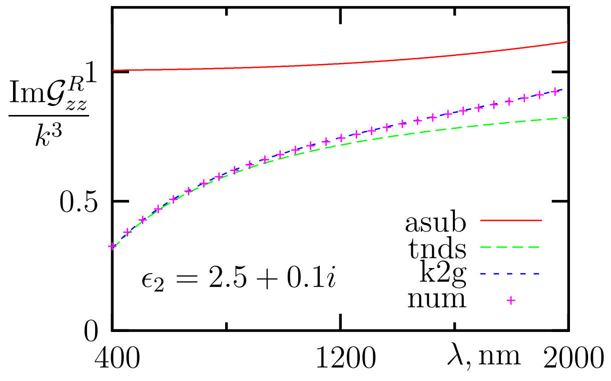

Figure 2. Example of using `GF_k2g.f`; $k = k_1 = 2\pi c/\lambda$.

8. IAQ (Infrequently Asked Questions)

- Why do the programs `GF_sde_*.f` compute only the reflected part of the Green's tensor while the programs `GF_num_*.f` can compute both the reflected and the total Green's tensor?

  The programs `GF_num_*.f` do not use any approximations (other than discretization of integrals) and, therefore, the reflected Green's tensor, $\mathcal{G}^R$, the free-space Green's tensor, $\mathcal{G}^F$ and the total Green's tensor, $\mathcal{G}^T$, which is the sum of the former two, are computed to the same level of approximation (in this case, no approximation at all). However, the programs `GF_sde_*.f` all use various levels of approximation to compute $\mathcal{G}^R$. It is not clear whether we should add this result to the *exact* $\mathcal{G}^F$ (which is easily computable) or to its expansion (Eq.(9) in the paper), computed to the same order as was used for $\mathcal{G}^R$. The choice may depend on the application and we are leaving it to the user.

- Are the programs optimized for speed? Can the numerical integration run faster?

  The programs have been optimized as much as is allowed by their fairly generic structure. In the majority of applications, execution time should not be a problem. Numerical integration, for example, should take less than a minute for something like 100 different wavelengths. Depending on the computer speed and the compiler used, one can even expect a few seconds. The analytical expansion programs run in less than noticeable time. However, it is not difficult to envision situations when the programs are converted to subroutines and used in nested loops. Also, if a metallic substrate with very low losses is used, convergence of numerical integrals may require a very large number of integration points. So, at least theoretically, execution speed may be a concern. Further optimization is quite possible, for example, by precomputing tables of Bessel functions, square roots and exponentials. However, we'd need to know the details: available memory, which parameters change from one run to another, etc. If you have a worthy scientific application and would like to talk to us about optimization, send us an e-mail.

- Can't the programs be converted to subroutines?

  Quite easily. But again, this depends on the application. Let us know if you need help.

- Why are there columns of zeros in the output files `K**_tsub`?

Because the real parts of all tensor coefficients $\mathcal{K}^{(l)}$ with $l > 3$ can not be computed and are, therefore, set to zero. The columns of zeros could certainly be omitted from the output but are kept in case we ever derive higher-order corrections to the real parts as well. Then modification of the code would be easier for us. The current type of output also simplifies book-keeping.

- Why the input parameters `N1`, `N2` and `N3` are declared as `INTEGER*8`?

  In case someone fancies using more than 2147483647 integration points.

- Why are there two different programs for a transparent substrate, `GF_sde_tnds.f` and `GF_sde_tdss.f`?

  This is an instance of the so-called optimization, most likely, unnecessary.
- Should I use `GF_num_exb.f` or `GF_num_inb.f`?

  `GF_num_exb.f` is almost always preferable. `GF_num_inb.f` is provided for comparison in case there is doubt.

- Which software I should use to visualize the data?

  Gnuplot will work best. Besides, it's free and available for both UNIX and Windows.
- Have the programs been tested?

  Yes, but only on one platform. We have compiled and tested the codes on an HP zx6000 Itanium-2 workstation running RedHat LINUX. We have used both the g77 GNU compiler and Intel's ifort compiler.